ATARI
serie XL

Invito alla Programmazione in BASIC con il Computer ATARI

a cura di: ATARI Software Support Group

ATARI
serie XL

Invito alla Programmazione in BASIC con il Computer ATARI

a cura di: ATARI Software Support Group

JACOPO CASTELFRANCHI EDITORE
Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

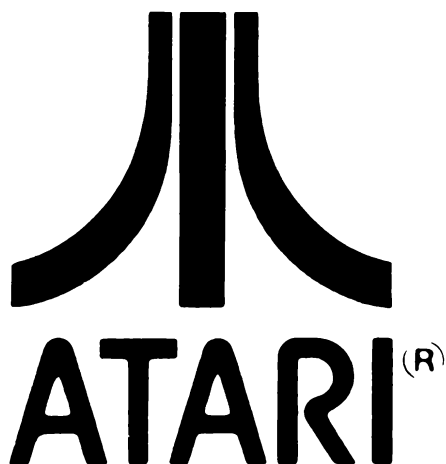
TABELLA DEI CODICI DI ERRORE

Questa tabella descrive in maniera sintetica i codici di errore riportati più ampiamente nell'appendice B

CODICE DI ERRORE	MESSAGGIO RELATIVO
2	Memoria insufficiente
3	Valore inaccettabile
4	Troppe variabili
5	Errore di lunghezza di una stringa
6	Dati insufficienti
7	Numero maggiore di 32767
8	Errore nell'istruzione di ingresso
9	Errore nel dimensionamento di una stringa o di un vettore
10	Trabocco della pila degli argomenti
11	Trabocco nel calcolo con virgola mobile
12	Riga non trovata
13	Manca la corrispondente istruzione FOR
14	Riga troppo lunga
15	Le istruzioni contenenti GOSUB o FOR sono scomparse
16	Errore nel RETURN
17	Errore nella pulizia interna
18	Carattere inammissibile in una stringa

Nota: Gli errori che seguono sono errori di INPUT/OUTPUT che derivano dall'uso delle unità disco, delle stampanti o di altre unità periferiche. Ulteriori informazioni relative a questi errori vengono fornite nei manuali o nei fogli di istruzione dei componenti in oggetto.

CODICE DI ERRORE	MESSAGGIO RELATIVO
19	Il programma da caricare è troppo lungo
20	Il numero di unità pariferica è troppo alto
21	Errore nel caricamento del file
128	L'operazione è stata interrotta perché è stato premuto il tasto **[BREAK]**.
129	IOCB
130	L'unità in questione non esiste
131	IOCB capace solo di scrivere
133	Unità o file non aperto
134	Il numero di IOCB è scorretto
135	IOCB capace solo di leggere
136	Fine del file
137	Record troncato
138	È stato superato il tempo di attesa concesso all'unità periferica
139	L'unità periferica non risponde
140	Bus seriale
141	Il cursore è fuori dei margini
142	Il bus seriale dei dati è sovraccarico
143	Errore di parità nel bus seriale dei dati
144	Errore segnalato dall'unità periferica
145	Errore nel confronto tra lettura e scrittura
146	Funzione non ancora costruita
147	La memoria RAM è insufficiente
160	Errore nel numero di unità disco
161	Troppi files aperti
162	Disco pieno
163	Errore di sistema nell'I/O non recuperabile
164	Il numero di file non corrisponde
165	Errore nel nome del file
166	Errore di valore nei dati di POINT
167	File protetto
168	Comando illegale
169	Direttorio pieno
170	File non trovato
171	POINT non valido

MANUALE DI RIFERIMENTO DEL LINGUAGGIO BASIC PER L'ELABORATORE PERSONALE ATARI

Abbiamo fatto il possibile per assicurare che questo manuale documenti accuratamente il funzionamento dell' ATARI XL. Tuttavia, poiché l'elaboratore viene continuamente migliorato ed aggiornato, non possiamo garantire l'accuratezza di quanto è stampato in questo manuale per molto tempo dopo la data di pubblicazione, né possiamo assumerci alcuna responsabilità per eventuali errori ed omissioni. Se necessario, saranno pubblicate versioni rivedute del manuale ed opuscoli aggiornati, che potranno essere acquistati scrivendo a:

Atari Software Support Group
P.O. Box 427
Sunnyvale, CA 94086

INDICE GENERALE

PREFAZIONE

Questo manuale, dando per scontato che l'utente abbia letto il libro "Atari BASIC - Guida autodidattica" o un altro testo introduttivo equivalente sul linguaggio BASIC, costituisce una guida di riferimento a programmi, istruzioni, funzioni ed applicazioni speciali del BASIC Atari.

Gli esempi di programmi riportati nel testo hanno la funzione di illustrare nel modo migliore il significato di tutte le funzioni e le parole chiave del linguaggio BASIC, e non intendono necessariamente rappresentare dei modelli di tecnica della programmazione.

Ogni capitolo contiene insiemi di comandi, funzioni o istruzioni che riguardano un aspetto particolare del BASIC Atari. Per esempio, il capitolo 9 contiene tutte le istruzioni sulle capacità grafiche che sono caratteristiche di Atari. Le appendici contengono dei brevi riferimenti ai termini, ai messaggi di errore, alle parole chiave del BASIC, agli indirizzi di memoria e all'insieme dei caratteri ATASCII.

Così come l'elaboratore personale Atari non è orientato ad una specifica applicazione, anche il manuale si rivolge ad un utente generico e ad applicazioni di tipo generale. L'Appendice H, infine, contiene dei programmi che illustrano alcune delle capacità del sistema ATARI.

CAPITOLO 1

INFORMAZIONI GENERALI

Questo capitolo illustra la terminologia del BASIC, le notazioni speciali, le abbreviazioni usate nel manuale e i tasti speciali della tastiera del Sistema di Elaborazione Personale ATARI 800XL. Rimanda gli altri capitoli di questo stesso manuale in cui i comandi e le istruzioni del linguaggio BASIC vengono trattati con riferimento ad applicazioni specifiche.

TERMINOLOGIA

BASIC: Beginner's All-purpose Symbolic Instruction Code (Codice simbolico per le istruzioni, adatto a tutti gli scopi, per principianti). È uno dei più semplici linguaggi di programmazione, creato presso l'università di Dartmouth nel 1964 e diventato uno standard di fatto per i microelaboratori personali verso la fine degli anni 70. Il BASIC ATARI comprende una parte comune a quasi tutti i linguaggi BASIC; si estende però in alcune funzioni per garantire la possibilità di sfruttamento delle caratteristiche particolari dell'elaboratore ATARI.

COMANDO BASIC: ordine di esecuzione immediata di un'azione. Comincia generalmente con una parola-chiave come LET, PRINT o RUN. Spesso un'espressione BASIC si presta ad essere utilizzata sia come ordine di esecuzione diretta di una azione (cioè come comando), sia come ordine da eseguire in modo differito (come istruzione); in questi casi, quando non vogliamo riferirci ad un particolare modo di esecuzione, ma soltanto, genericamente, all'espressione stessa, parleremo ancora di comando. Questo fatto non dovrebbe creare confusione, in quanto per ognuna delle parole chiave che verranno descritte in seguito specificheremo se esse possono entrare a costituire soltanto un comando (vero e proprio), o soltanto un'istruzione, oppure se si prestano ad essere utilizzate in entrambi i modi.

COSTANTE: valore espresso direttamente come numero anziché essere rappresentato da un nome di variabile. Per esempio, nell'espressione X = 100, X è un nome di variabile e 100 è una costante (vedi Variabile).

ESPRESSIONE: qualsiasi combinazione legale di variabili, costanti, operatori e funzioni usati insieme per calcolare un valore. Il valore dell'espressione risultante può essere un numero, un valore logico o una stringa. Conseguentemente l'espressione si dice di tipo numerico, logico o di tipo stringa.

FUNZIONE: sequenza di calcolo codificata nella memoria dell'elaboratore in modo da essere richiamabile dai programmi dell'utente.

Le funzioni non vanno confuse con le istruzioni: sono piuttosto dei sottoprogrammi invisibili all'utente, delle vere e proprie scatole nere, che il BASIC mette a disposizione dell'utente

perché, inserendole in un programma, gli consentono di calcolare dei valori che all'uscita del sottoprogramma vengono riconsegnati al programma principale.

Esempi di funzione sono RND (random, a caso), FRE (spazio di memoria libero), INT (approssimazione intera di un numero reale). In molti casi, il valore viene semplicemente assegnato ad una variabile, per un uso successivo. In altri casi il valore può apparire sullo schermo immediatamente. Il Capitolo 6 contiene maggiori informazioni sulle funzioni. Negli esempi che seguono mostriamo alcuni modi in cui le funzioni possono apparire nei programmi.

```
10 PRINT RND(0)
```

(stampa il numero casuale fornito da una procedura interna)

```
10 X=100+COS(45)
```

(aggiunge a 100 il valore calcolato dalla funzione coseno per l'argomento 45 e conserva il risultato nella variabile X)

ISTRUZIONE BASIC: ordine di esecuzione di un'azione in un momento differito. Le istruzioni sono formalmente identiche ai comandi ma devono essere scritte su una riga preceduta da un'etichetta numerica e possono estendersi oltre la riga fisica dello schermo, per tre righe consecutive che si dicono costituire una riga logica. L'etichetta fa capire al BASIC che si tratta di un'istruzione, ossia di un ordine che non deve essere eseguito subito, a differenza di quanto accade per i comandi.

NUMERO MASSIMO DI NOMI DELLE VARIABILI: Il BASIC Atari limita l'utente ad usare non più di 128 nomi di variabili. Per superare questo problema si possono tuttavia utilizzare gruppi di variabili con lo stesso nome, distinte tra loro da un indice numerico: si tratta della struttura di "vettore" di cui si parlerà più avanti.

Il BASIC conserva la "memoria" di una variabile anche se questa viene abolita da un programma: la variabile, in altri termini, rimane nella tabella dei nomi costruita dal BASIC (non visibile all'utente), e continua quindi ad occupare spazio di memoria. Se sullo schermo appare un ERRORE-4 (Troppe variabili), per far posto a nuovi nomi di variabili occorre procedere nel modo descritto qui espresso. Si devono dare i tre seguenti comandi.

```
LIST filespec
NEW
ENTER filespec
```

Dove filespec, come vedremo nel seguito di questo capitolo, rappresenta un nome legale di file (per il disco o per la cassetta).

Per capire cosa provocano le tre istruzioni date sopra occorre tener presente che i programmi risiedono in memoria non nella forma in cui appaiono scritti sullo schermo, ma in una forma codificata, più sintetica, in cui ad ogni ordine del BASIC ATARI corrisponde un codice convenzionale. Le parole che, non essendo riconosciute come parole chiave, vengono interpretate dal BASIC come nomi di variabili assegnate dall'utente, vengono anch'esse codificate. Questo codice viene ricavato mettendo le parole in una tabella ed utilizzando come codice il posto della parola stessa nella tabella. In questo modo quindi si realizza un risparmio di spazio di memoria proporzionale al numero delle occorrenze della variabile nel programma perché il codice assegnato dal BASIC alla variabile occupa sempre meno spazio di memoria del nome (esteso) della variabile stessa.

Il comando LIST filespec scrive una versione non codificata del programma su disco o cassetta. Il comando NEW cancella dalla memoria centrale il programma e la tabella dei simboli. Attraverso il comando ENTER filespec il programma viene quindi reintrodotto in memoria, subendo un nuovo procedimento di codifica che genera, al tempo stesso, una nuova tabella dei nomi delle variabili.

OPERATORE: chiamiamo operatori un gruppo di simboli che rappresentano operazioni o relazioni aritmetiche o logiche, i quali vengono usati per formare le cosiddette espressioni del BASIC. Gli operatori comprendono gli operatori aritmetici di addizione (+), di sottrazione (—), di moltiplicazione (*), di divisione (/), di elevazione a potenza (∧); gli operatori logici di relazione come maggiore di (>), minore di (<), uguale a (=), maggiore o uguale a (>=), minore o uguale a (<=) e non uguale (<>). Sono operatori logici anche i simboli AND, NOT e OR. Gli operatori + e — possono essere usati anche come operatori unari, ossia ad un solo argomento; per esempio, –3. Non bisogna porre mai due operatori unari uno appresso all'altro, per esempio – –3, perché l'elaboratore non è capace di interpretarli in modo corretto.

PAROLA CHIAVE BASIC: una parola chiave è una successione di caratteri che ha un significato operativo "convenzionale" per il linguaggio BASIC. Le parole chiave, come abbiamo già accennato a proposito del procedimento per ridurre la tabella delle varabili, vengono tradotte dal BASIC ATARI in simboli interni (attraverso un codice), allo scopo di ridurre l'occupazione di memoria dei programmi. Le parole chiave servono, insieme con altre parole convenzionali scelte dall'utente per rappresentare le variabili, a comporre le istruzioni o i comandi BASIC. A volte si parla anche di "parole riservate" perché le parole chiave non possono essere usate con altri significati che quelli convenzionali assegnati dal BASIC, (l'Appendice A contiene l'elenco di tutte le parole chiave del linguaggio BASIC disponibili su Atari).

PROGAMMA: una successione di istruzioni BASIC da eseguire in modo differito.

RIGA FISICA: una riga di caratteri così come appare sullo schermo televisivo.

RIGA LOGICA: Ogni riga numerata in un programma BASIC, quando appare sullo schermo, individua una riga logica, che può essere costituita da un minimo di una ad un massimo di 3 righe fisiche. Quando si scrive una riga logica il BASIC interrompe automaticamente la scrittura se viene raggiunto il limite massimo della riga logica stessa, oppure non appena viene premuto il tasto **[RETURN]**. Quando nell'immissione di una riga logica più lunga della riga fisica, si raggiunge il limite di quest'ultima, il cursore si sposta automaticamente all'inizio della riga fisica successiva. Se non viene premuto il tasto **[RETURN]** le due righe fisiche apparterranno alla stessa riga logica.

STRINGA: una stringa è costituita da una serie di caratteri racchiusi tra virgolette. "ABRACADABRA" è una stringa. Sono stringhe anche "ATARI COSTRUISCE DEI FANTASTICI ELABORATORI" e "123456789". Una stringa è molto simile ad una costante, in quanto anch'essa può essere conservata in una variabile. I nomi di variabili che contengono stringhe differiscono da quelli delle variabili che contengono numeri per il fatto di terminare con il carattere speciale "$". La stringa "ATARI 800", per esempio, può essere assegnata ad una variabile chiamata A$, usando LET (che è opzionale), come negli esempi che seguono:

```
10 LET A$ = "ATARI 800"
```

(Notare le virgolette).

Oppure...

10 A$ = “ATARI 800”

(In questa istruzione la parola chiave LET è opzionale, le virgolette sono obbligatorie).

Una stringa non può contenere virgolette al suo interno: questo fatto non può sorprendere se si pensa che le virgolette sono necessarie per delimitare le stringhe. La chiusura delle virgolette può essere omessa se (e solo se) esse costituiscono l'ultimo carattere di una riga logica.

STRINGA DI COMANDI O ISTRUZIONI: successione di comandi o istruzioni su una stessa riga logica separati dal segno dei due punti. Una stringa di comandi può essere usata in modo diretto, per dare più comandi di seguito, o in modo differito, per scrivere più istruzioni su una stessa riga, riducendo lo spazio di memoria occupato dal programma. In quest'ultimo caso parleremo di stringa di istruzioni.

VARIABILE: sta ad indicare un valore (numerico, logico o stringa) conservato in una posizione di memoria associata ad un nome convenzionale, scelto dall'utente. Il termine “variabile” esprime il fatto che il valore associato al nome può essere sostituito nel corso dell'elaborazione o dell'interazione con l'elaboratore. Il nome della variabile può essere lungo fino a 120 caratteri, deve cominciare con una lettera alfabetica e può contenere solo lettere maiuscole o cifre numeriche. Nella scelta dei nomi delle variabili occorre evitare l'uso delle parole chiave (anche soltanto all'inizio del nome della variabile). Tali nomi potrebbero non essere riconosciuti come nomi di variabili e dare luogo ad interpretazioni scorrette. Vediamo alcuni esempi di assegnazione di valori a delle variabili.

LET C12DVB=1.234
LET VARIABILE 112=267.543
LET A=1
LET F5TH=6.5
LET THISND = 59.809

(Anche qui il LET è opzionale e può essere omesso).

Aggiungiamo degli esempi scorretti con una stella davanti:

*LET COS=5

(La variabile indicata con COS si confonde con la parola chiave che il BASIC riserva per la funzione coseno).

*LET CONT=3

(La variabile indicata con CONT si confonde con la parola chiave CONT usata dal BASIC per continuare l'esecuzione di un programma).

*LET NEWYORK=“CITTÀ”

(La variabile indicata con NEWYORK si confonde con la parole chiave NEW usata dal BASIC per cancellare il programma conservato nella memoria centrale).

VETTORI E VARIABILI VETTORE: un vettore si può definire come una lista di valori associati ad un unico nome simbolico chiamato variabile (di tipo) vettore o variabile vettoriale; alternativamente si può dire che un vettore rappresenta una successione di posizioni di memoria in cui possono essere archiviati dati dotati di qualche omogeneità ed associati al nome della variabile dal quale vengono differenziati mediante i valori di un indice numerico. Ogni valore (ogni posizione) del vettore si chiama “elemento” del vettore stesso. Supponiamo che il vettore di nome A sia dotato di 7 elementi. Ci si riferisce a questi

elementi usando il nome della variabile vettoriale seguito dall'indice dell'elemento, racchiuso tra parentesi tonde: A(2), A(3), A(4), etc. L'espressione che identifica un elemento si dice anche variabile indicizzata, oppure variabile sottoscritta, in memoria del fatto che in matematica l'indice viene scritto mezza riga al di sotto. In ogni elemento del vettore può dunque essere conservato un numero. L'assegnazione dei valori agli elementi di un vettore può essere compiuta soltanto elemento per elemento, usando ripetutamente l'istruzione LET, o mediante un ciclo di programma (mediante le istruzioni FOR... NEXT, vedi capitolo 8).

Nota: quando si scrive una variabile sottoscritta per indicare un elemento di un vettore occorre fare bene attenzione a non lasciare spazi bianchi tra il nome della variabile e le parentesi che racchiudono l'indice dell'elemento del vettore. Facciamo degli esempi:

Corretto	Scorretto
A(23)	A (23)
ARRAY(3)	ARRAY (3)
X123(38)	X123 (38)

NOTAZIONI SPECIALI

Per spiegare la struttura dei comandi e delle espressioni BASIC nel seguito di questo manuale useremo, come si fa spesso, dei simboli per indicare certi tipi di oggetti. Dal momento che per ogni tipo di oggetto è conveniente usare, per quanto è possibile, sempre lo stesso simbolo, vale la pena spiegare i simboli una volta per tutte all'inizio.

FORMATO DI UNA RIGA DI PROGRAMMA: si parlerà spesso, nel seguito, di formati, per indicare la struttura di un'espressione BASIC composta da parole chiave mescolate ai simboli che indicano altri tipi di oggetti più elementari. Ogni riga di un programma BASIC deve contenere un numero di riga (abbreviato in lineno) all'inizio della riga stessa, seguito da una parola chiave, seguita a sua volta dal corpo dell'istruzione ed infine da un comando che termina la riga (il tasto **[RETURN]**). Vediamo i quattro componenti di una riga in un esempio concreto:

	ISTRUZIONE		
Numero di riga	Parola-chiave	Corpo	Terminatore
100	PRINT	A/X * (Z+4,567)	**[RETURN]**

Su una stessa riga si possono scrivere diverse istruzioni, a patto di separarle mediante il segno di due punti (:). Vedi IF/THEN nel capitolo 5, e il capitolo 11.

LETTERE MAIUSCOLE: per tutto il corso del manuale scriveremo in maiuscolo le parole chiave, le quali, per essere riconosciute dal BASIC come tali, devono essere composte mediante lettere maiuscole, esattamente come appaiono nel testo. I caratteri inversi sul video (nero su bianco anziché bianco su nero) non vengono accettati dal BASIC se non nel caso del comando RUN. Ecco alcuni esempi di parole chiave scritte in maiuscolo.

PRINT INPUT LIST END GOTO GOSUB FOR NEXT IF

LETTERE MINUSCOLE: useremo le minuscole, per tutto il corso del manuale, per indicare vari tipi di oggetti utilizzabili in un programma, come le variabili (var), le espressioni (exp), e simili. Le abbreviazioni usate per questi tipi di oggetti sono riportate nel prossimo paragrafo.

OGGETTI TRA PARENTESI QUADRE: Le parentesi quadre [ ], contengono oggetti che possono, ma non necessariamente devono, apparire in quel contesto. Quando l'oggetto racchiuso tra parentesi è seguito da tre puntini [exp,...], si intende che in quella posizione può comparire un numero qualunque di oggetti di quel tipo (eventualmente zero).

OGGETTI TRA PARENTESI GRAFFE: nella spiegazione di alcune istruzioni useremo un insieme di oggetti disposti in una pila tra parentesi graffe per indicare che in quel contesto può apparire uno (ed esattamente uno) degli oggetti disposti nella pila. Nell'esempio che segue si può scrivere GOTO oppure GOSUB, ma non entrambi.

$$100 \left\{ \begin{array}{c} \text{GOTO} \\ \text{GOSUB} \end{array} \right\} 2000$$

ABBREVIAZIONE DELLE PAROLE CHIAVE: alcune parole chiave possono essere abbreviate. Ogni volta che è ammessa, indicheremo l'abbreviazione tra parentesi tonde dopo la parola chiave; ad es., LET (L.).

ABBREVIAZIONI

Nella lista che segue sono riportate le abbreviazioni che useremo sistematicamente nel manuale per indicare oggetti di certi determinati tipi. Accanto all'abbreviazione viene data una spiegazione intuitiva, non formale, del tipo di oggetto a cui ci si riferisce.

avar = variabile aritmetica. Una variabile cui è assegnato un valore numerico. Il nome di una variabile aritmetica può consistere di un minimo di 1 fino ad un massimo di 120 caratteri alfanumerici, ma deve cominciare con un carattere alfabetico e gli altri eventuali caratteri alfabetici devono essere esclusivamente maiuscoli e non video inversi.

svar = variabile stringa. Una variabile cui è assegnata come valore una stringa di caratteri. Il nome di una variabile stringa deve essere composto secondo le stesse regole delle variabile aritmetiche, ma inoltre il nome di una svar deve terminare con il carattere "$". Le variabili stringa possono essere sottoscritte. Vedi Capitolo 7, **STRINGHE.**

mvar = variabile matrice, detta anche variabile indicizzata o sottoscritta. Elemento di un vettore o di una matrice. Il nome della variabile per il vettore o per la matrice, considerata come un tutto, può essere un qualsiasi nome di variabile legittimo, come A, X, Y, ZIP o K. La variabile sottoscritta (nome per l'elemento particolare) comincia con la variabile matrice, usa poi un numero, una variabile o un'espressione tra parentesi, che segue immediatamente la variabile vettore o la variabile matrice. Per es., A(ROW), A(1), A(X+1).

var = variabile. Indica una variabile qualsiasi. Può essere usata al posto di mvar, avar o svar.

aop = operatore aritmetico.

lop = operatore logico.

aexp = espressione aritmetica. Generalmente composta da una variabile, una funzione, una costante o da due espressioni aritmetiche separate da un operatore aritmetico.

lexp = espressione logica. È una espressione composta di due espressioni aritmetiche o stringhe separate da un operatore logico. Si dice che una espressione logica ha un valore che può essere 1 (nel caso in cui l'espressione risulta vera) o 0 (nel caso in cui l'espressione è falsa).

Per esempio, l'espressione 1 < 2 da valore 1 (vero), mentre l'espressione "LIMONE" = "ARANCIA" dà come valore 0 (falso) poiché le due stringhe non sono uguali.

sexp = espressione stringa. Una espressione stringa può consistere di una variabile stringa, di una costante, o di una funzione il cui risultato è il valore di una stringa.

exp = espressione qualsiasi, sexp o aexp.

lineno = numero di riga. È una costante che individua una particolare riga di un programma BASIC. Può essere un numero intero da 0 a 23767. La numerazione delle righe determina l'ordine di esecuzione delle stesse nel programma.

adata = dato ATASCII. Un qualsiasi carattere ATASCII, escluse virgole e ritorni di carrello (vedi Appendice C).

filespec = specificazione di file. Un'espressione stringa che si riferisce ad un dispositivo come la tastiera o a un file di disco. Contiene: informazioni sul tipo di dispositivo di ingresso/uscita dei dati, il numero di dispositivo, i due punti (:), il nome di un file ed una estensione opzionale del nome del file (vedi istruzione OPEN, Capitolo 5).

Esempio di filespec: "D1 : MENO MAL".

MODI DI FUNZIONAMENTO DELL' ELABORATORE

MODO DIRETTO: non usa numeri di riga ed esegue il comando immediatamente dopo che è stato premuto il tasto **[RETURN]**.

MODO DIFFERITO: usa i numeri di riga e rimanda l'esecuzione delle istruzioni finché non viene dato il comando RUN.

MODO ESECUTIVO: detto anche, talvolta, modo RUN. Una volta dato il comando RUN, viene trattata ed eseguita ogni riga di programma.

MODO MEMO PAD (TACQUINO DI APPUNTI): è un modo di usare l'elaboratore senza programmi che permette all'utente di eseguire prove con la tastiera oppure di lasciare messaggi sullo schermo. Nessuno dei caratteri scritti in questo modo ha alcun effetto sul programma contenuto in memoria.

TASTI PER FUNZIONI SPECIALI

Tasto di inversione del video, o **"ATARI INVERSE KEY".** Questo tasto fa sì che, sullo schermo, il testo da chiaro diventi scuro e che venga anche invertito il colore dello sfondo, da nero in bianco. Per tornare alla situazione normale premere il tasto una seconda volta.

[RETURN] Tasto di ritorno carrello. È il tasto che serve ad indicare all'elaboratore che è stata terminata una riga di BASIC. Quando si preme il tasto **[RETURN]** dopo avere scritto una o più istruzioni su una riga numerata, il BASIC interpreta la riga e la traduce nel suo codice interno, aggiungendola alle righe già composte del programma residente nella memoria RAM.

[CAPS/LOWR] Tasto per il controllo delle maiuscole e delle minuscole. Normalmente (appena acceso) l'elaboratore scrive sullo schermo caratteri soltanto maiuscoli. Se si preme

il tasto **[CAPS/LOWR]** si passa dal modo maiuscolo al modo misto (caratteri minuscoli oppure maiuscoli se, assieme al tasto del carattere, si preme simultaneamente anche il tasto **[SHIFT]**). Per tornare alla situazione ordinaria (sole maiuscole) occorre premere simultaneamente il tasto **[SHIFT]** insieme con il tasto **[CAPS/LOWR]**.

[BREAK] Se durante l'esecuzione di un programma si preme il tasto **[BREAK]** l'esecuzione si arresta: può essere ripresa scrivendo il comando CONT e premendo quin '' il tasto **[RETURN]**.

[RESET] Tasto di riposizionamento del sistema. Premendo il tasto **[RESET]** durante l'esecuzione di un programma si ottiene un effetto simile a quello di **[BREAK]** nel senso che l'esecuzione si arresta immediatamente. Tuttavia con il tasto **[RESET]** si ottengono anche altri effetti: lo schermo ritorna nella modalità grafica 0 e viene completamente cancellato; inoltre i margini e tutti gli altri valori relativi allo schermo vengono riportati ai relativi valori per difetto (quelli che valgono al momento dell'accensione dell'elaboratore).

[SET-CLR-TAB] Tasto di controllo del tabulatore. Si chiama tabulatore un meccanismo (simile a quello disponibile su ogni macchina da scrivere) che consente di fissare delle posizioni (o valori) sulla riga in maniera tale che, premendo un certo tasto, si può far avanzare il cursore velocemente dalla posizione attuale alla posizione prefissata più vicina sulla destra. **[SET-CLR-TAB]** è il tasto della tastiera ATARI che fa avanzare il tabulatore. I valori del tabulatore possono essere modificati: si possono cioè aggiungere nuove posizioni o cancellare posizioni esistenti. Per inserire un nuovo valore al tabulatore occorre portare il cursore nella posizione prestabilita. Quindi si devono premere simultaneamente i tasti **[SHIFT]** e **[SET-CLR-TAB]**. Per cancellare un valore del tabulatore occorre premere simultaneamente i tasti **[CTRL]** e **[SET-CLR-TAB]**. I valori del tabulatore possono essere alterati anche in modo differito, ossia durante l'esecuzione di un programma, mediante opportune istruzioni. Tali istruzioni si ottengono dai comandi corrispondenti illustrati sopra, facendo precedere il comando dal numero di riga, seguito dalla parola chiave PRINT, seguita dalle virgolette a dal tasto **[ESC]**, come negli esempi seguenti.

Esempi:

```
100 PRINT "[ESC] [SHIFT] [SET-CLR-TAB]"
200 PRINT "[ESC] [CTRL] [SET-CRL-TAB]"
```

Quando si accende l'elaboratore o quando si preme il tasto **[RESET]**, al tabulatore vengono assegnati dei valori che diciamo valori per difetto, che corrispondono alle colonne 7, 15, 23, 31 e 39.

[INSERT] Tasto per inserire caratteri o righe. Si può correggere una parola, un testo, un programma, inserendo nuovi caratteri o un'intera riga. Per inserire un carattere, dopo aver portato il cursore nel punto in cui si vuole inserire il carattere, si deve premere il tasto **[INSERT]** assieme a **[CTRL]** e poi il tasto del carattere da inserire. Se si vuole inserire un'intera riga, si devono premere simultaneamente i tasti **[SHIFT]** ed **[INSERT]** seguiti dalla riga che si vuole inserire.

[DELETE BACK S] Tasto per tornare indietro cancellando. Si può tornare indietro di uno spazio cancellando il carattere appena battuto. Battendo il tasto **[DELETE BACK S]** il carattere alla sinistra del cursore viene sostituito con uno spazio mentre il cursore torna indietro di una posizione.

[DELETE BACK S] Cancellazione in avanti. Si può cancellare un carattere in avanti premendo simultaneamente i tasti **[CTRL]** e **[DELETE]**. Si può anche cancellare un'intera riga premendo simultaneamente i stati **[SHIFT]** e **[DELETE]**.

[CLEAR] Tasto di cancellazione totale. Il tasto **[CLEAR]** premuto contemporaneamente ai tasti **[SHIFT]** e **[CTRL]** cancella tutto quello che si trova sullo schermo e porta il cursore nell'angolo superiore sinistro dello schermo stesso.

[ESC] Tasto Escape. Permette di includere in un programma un comando abitualmente usato in modo diretto, che dunque non verrà eseguito immediatamente ma solo dopo il comando RUN.

Esempio: per cancellare lo schermo, potete immettere nel vostro programma le seguenti istruzioni:

10 PRINT "**[ESC] [CTRL] [CLEAR]**"

Il tasto **[ESC]** può anche essere usato, in congiunzione con altri tasti, per ottenere sullo schermo alcuni simboli grafici particolari. L'elenco di questi simboli, assieme ai caratteri da battere in congiunzione con il tasto **[ESC]** per ottenerli, si trova nell'Appendice F o sul retrocopertina.

OPERATORI ARITMETICI

Il sistema di Elaborazione Personale ATARI usa cinque operatori aritmetici, rappresentati dai seguenti simboli.

+ addizione (di solito binario, può essere anche usato come operatore unario, come nell'espressione +5).
— sottrazione (di solito binario, ma anche unario, come nell'espressione —5).
* moltiplicazione.
/ divisione.
∧ elevazione a potenza.

OPERATORI LOGICI

Un operatore logico è un operatore che produce il valore vero o il valore falso a partire da un certo numero di espressioni dello stesso tipo. Il valore **vero** è rappresentato dal numero intero +1, il valore **falso** è rappresentato dal numero intero 0.

Ci sono operatori logici di due tipi: unari e binari. Gli operatori unari sono quelli che si applicano ad una sola espressione di tipo logico per produrre il valore vero o falso. Gli operatori binari sono quelli che si applicano a due espressioni dello stesso tipo per produrre il valore vero o falso. L'unico operatore unario è l'operatore **NOT.** Gli operatori binari sono:

AND E logico.
OR O logico.

Esempi:

10 IF A=12 AND T=0 THEN PRINT "BENE"

La parola BENE viene stampata se e solo se entrambe le espressioni A=12 e T=0 sono vere nel momento in cui l'istruzione viene eseguita.

10 A = (C>1) AND (N<1)

Se entrambe le espressioni C>1 e N<1 sono vere la variabile A viene posta uguale a +1, altrimenti viene posta uguale a 0.

L'elaboratore considera false (=0) le espressioni numeriche con valore 0 e vere (=1) tutte le espressioni numeriche con valore diverso da 0 (valori positivi o negativi, anche non interi).

```
10 A = (C+1) OR (N-1)
```

Se almeno una delle due espressioni C+1 o N-1 è diversa da 0, alla variabile A viene assegnato il valore +1, altrimenti il valore 0.

```
10 A = NOT (C+1)
```

Se l'espressione C+1 è uguale a 0 (ossia è falsa) allora la variabile A riceve il valore +1, altrimenti riceve il valore 0.

Il resto degli operatori binari sono relazionali.

- < minore. Produce il valore vero se la prima espressione è minore alla seconda.
- \> maggiore.Produce il valore vero se la prima espressione è maggiore della seconda.
- = uguale. Produce il valore vero se le due espressioni sono uguali.
- <= minore o uguale. Produce il valore vero se la prima espressione è minore o uguale alla seconda.
- \>= maggiore o uguale. Produce il valore vero se la prima espressione è maggiore o uguale alla seconda.
- <> diverso da. Produce il valore vero se le due espressioni non sono uguali.

Questi operatori si possono usare sia all'interno delle istruzioni IF/THEN, sia per costruire delle variabili logiche.

REGOLE DI PRECEDENZA TRA OPERATORI

Le parentesi, come è noto, servono per indicare l'ordine in cui devono essere eseguite le operazioni contenute in una determinata espressione. Tale ordine, come è noto, è rilevante, nel senso che cambiando l'ordine delle operazioni si possono ottenere risultati diversi. Più le parentesi sono interne e prima le operazioni in esse contenute vanno eseguite. Il BASIC riconosce le parentesi e le rispetta come segni di precedenza. Tuttavia in alcune espressioni si possono inserire più operatori di seguito senza racchiuderli tra parentesi. In questo caso l'ordine in cui le operazioni vengono eseguite non è sempre quello di scrittura (da sinistra a destra).

Esistono delle regole di precedenza che il linguaggio BASIC rispetta e che consentono di fare a meno di molte parentesi.

In generale si eseguono per prime le operazioni nell'insieme di parentesi più interno, si procede poi verso l'esterno affrontando le operazioni nei livelli immediatamente successivi. Si dicono "nested", inseriti l'una nell'altra, insiemi di parentesi racchiuse in un altro insieme. Le operazioni sullo stesso livello di parentesi vengono eseguite nel seguente ordine ordine di precedenza (l'elenco inizia dalle operazioni aventi ordine di precedenza più alta e finisce con quelle aventi ordine di precedenza più bassa).

<, >, <=, >=, <> Operatori relazionali usati in espressioni stringa. Hanno tutti lo stesso ordine di precedenza e sono eseguiti da sinistra a destra.

– meno unario.

*,/ moltiplicazione e divisione hanno lo stesso ordine di precedenza e vengono eseguite da sinistra a destra.

+,— addizione e sottrazione hanno lo stesso ordine di precedenza e vengono eseguite da sinistra a destra.

<,>,=,<=,>=,<> operatori relazionali in espressioni numeriche: hanno lo stesso livello di precedenza e vengono eseguiti da sinistra a destra.

NOT operatore unario di negazione.

AND E logico.

OR O logico.

FUNZIONI INTERNE

Il capitolo 6 (**BIBLIOTECA DI FUNZIONI**) spiega il modo di operare delle funzioni aritmetiche e delle funzioni speciali incorporate nel BASIC Atari.

GRAFICA

Chiamiamo grafica la capacità dell'elaboratore di esprimersi in forma grafica.

La grafica dell'ATARI prevede ben 9 livelli o modalità diverse. I comandi grafici sono stati studiati per permettere la massima flessibilità nella scelta del colore e del metodo di costruzione dei disegni. Il capitolo 9 fornisce spiegazioni su ogni comando e numerosi esempi sull'uso degli stessi.

SUONI E DISPOSITIVI DI CONTROLLO

L'elaboratore personale Atari è capace di emettere una grande varietà di suoni, tra cui esplosioni simulate, musica elettronica ed altri segnali. Il capitolo 10 illustra il comando SOUND ed i vari dispositivi di controllo usati per i giochi (controllori a leva, manopola o a tastiera, etc.).

SCHERMO CIRCOLARE E TASTIERA AUTORIPETITIVA

Gli elaboratori personali ATARI sono dotati di uno schermo che diciamo "circolare" perché quando il cursore supera il margine destro viene automaticamente riportato all'inizio della riga successiva, sul margine sinistro. Questo consente di scrivere sullo schermo con maggiore rapidità perché permette di non prestare attenzione alla fine della riga.
La tastiera dell'ATARI è capace di ricordare un carattere in più di quelli già visualizzati sullo schermo, consentendo così a chi batte di regolare la sua velocità senza perdere delle battute.
La tastiera dell'ATARI, inoltre, è dotata della capacità di autoripetizione: quando si tiene premuto un tasto per più di mezzo secondo, il carattere battuto viene automaticamente ripetuto sullo schermo sino a che non si abbandona il tasto.

MESSAGGI

Se si commettono errori nell'introduzione dei dati, sullo schermo appare la riga appena battuta preceduta dal messaggio ERROR—, con il carattere che ha causato l'errore illuminato in modo diverso dagli altri. Dopo aver corretto il carattere sulla riga originale, dovete cancellare la riga che contiene ERROR– prima di premere **[RETURN]**. L'appendice B contiene una lista di tutti i messaggi di errore e le relative spiegazioni.

COMANDI

Ogni volta che il cursore (□) appare sullo schermo, l'elaboratore è pronto ad accettare informazioni in entrata. Scrivete il vostro COMANDO (diretto o differito) e premete il tasto **[RETURN]**. In questo capitolo descriveremo i comandi che permettono di cancellare ogni informazione presente nella memoria dell'elaboratore ed altri utili comandi di controllo.

Ricordiamo che abbiamo convenuto di chiamare comandi gli ordini eseguibili in modo diretto ed istruzioni gli ordini da eseguire in modo differito. Parleremo però egualmente di comandi quando non intendiamo specificare se si tratta di un comando vero e proprio o di una istruzione, oppure quando ci vogliamo riferire in generale ad un'espressione BASIC che può essere utilizzata sia come comando che come istruzione.

In questo capitolo illustreremo i comandi costruiti mediante le seguenti parole chiave:

BYE
CONT
END
LET
LIST
NEW
REM
RUN
STOP

Avvertiamo anche che, per semplicità, in questo capitolo e in quelli seguenti useremo spesso i termini comando ed istruzione laddove, per correttezza, si dovrebbe usare il termine parola chiave. Le parole chiave sono infatti gli ingredienti dei comandi, i quali molto spesso contengono, oltre a quelle, altri ingredienti. Tuttavia questo modo di esprimersi non è di solito fonte di equivoci, perché il contesto è sufficiente a chiarire completamente i significati.

BYE (B.)

Formato: BYE
Esempio: BYE

La funzione del comando BYE è quella di uscire dal BASIC e di portare l'elaboratore nel sistema di autodiagnosi. Per tornare di nuovo al BASIC occorre premere il tasto **[RESET]**

CONT (CON.)

Formato: CONT
Esempio: CONT

Questo comando, seguito dal tasto **[RETURN]**, permette di riprendere l'esecuzione di un programma precedentemente interrotta. Ci sono tre modi per interrompere l'esecuzione di un programma. Il primo è esterno: si può interrompere in qualsiasi momento l'esecuzione di un programma premendo il tasto **[BREAK]**. Gli altri due modi sono interni, cioè vengono codificati nel programma stesso, e si realizzano con l'istruzione STOP o l'istruzione END. In tutti e tre i casi l'esecuzione è bloccata fino al momento in cui viene dato il comando CONT (seguito, come al solito dal tasto **[RETURN]**), con il quale essa riprende a partire dal numero di riga che, nella successione delle istruzioni che costituiscono il programma, segue immediatamente l'istruzione alla quale l'esecuzione del programma era stata interrotta.

Nota: Se l'istruzione alla quale si interrompe l'esecuzione del programma si trova in una riga logica in cui appaiono altre istruzioni che non siano già state eseguite prima dell'interruzione, le istruzioni residue non verranno più eseguite. CONT riprende l'esecuzione a partire dalla riga numerata successiva. Se un programma viene bloccato prima che sia stata completata l'esecuzione di un ciclo, l'esecuzione dello stesso ciclo può risultare scorretta.

Il comando CONT non può essere inserito in un programma (non può essere cioè eseguito in modo differito).

END

Formato: END
Esempio: 1000 END

Il comando END si può dare sia in modo diretto che in modo differito. In modo differito, serve ad interrompere l'esecuzione di un programma. (Si noti tuttavia che nel BASIC Atari non è strettamente necessario porre un'istruzione END alla fine del programma: dopo aver eseguito l'ultima istruzione, infatti, il BASIC Atari chiude automaticamente tutti i files ed interrompe i suoni eventualmente attivi). In modo diretto l'istruzione END può essere usata per chiudere i files ed per interrompere i suoni.

LET (LE.)

Formato: (LET) var = exp
Esempi: LET X = 3.142 * 16
LET X = 2

Questa istruzione serve per assegnare un valore ad una variabile, ma in essa la parola chiave LET non è strettamente necessaria: nel BASIC Atari si usa semplicemente per aggiungere chiarezza o per uniformità con altre versioni del BASIC. Nell'istruzione di assegnazione di un valore ad una variabile si può quindi omettere tranquillamente la parola chiave LET, e scrivere soltanto, per esempio, X = 354.

LIST (L.)

Formato:	LIST [lineno[, lineno]]
	LIST [filespec [,lineno [,lineno]]]
Esempi:	LIST
	LIST 10
	LIST 10,100
	LIST "P.",20,100
	LIST "P"
	LIST "D:DEMO.LST"

Questo comando, se non è seguito da un numero (o numeri) di riga, fa apparire sul video tutte le righe del programma attualmente in memoria; altrimenti, appariranno una o più righe specifiche. Per esempio il comando LIST 10,100 **[RETURN]** farà apparire sullo schermo le righe numerate da 10 a 100. Se, nello scrivere il programma, l'utente non ha scritto le righe in ordine crescente di lineno, l'elaboratore, eseguendo il comando LIST, le metterà atutomaticamente in ordine.

Scrivendo L."P", il programma residente in RAM viene scritto sulla stampante.

Potete usare il comando LIST anche in modo differito, ad esempio come parte di una routine per la ricerca di errori (vedi **TRAP** nel capitolo 4).

Il comando LIST serve inoltre a registrare i programmi su nastro o su disco. Si utilizza allora il secondo formato riportato sopra, quello in cui il comando LIST è seguito da una filespec e da numeri di riga (per ulteriori informazioni sulle unità periferiche, vedi il capitolo 5). Se si vuole riportare sul nastro l'intero programma non è necessario specificare i numeri di riga.

Esempi:	LIST "C1"
	1000 LIST "C1"

NEW

Formato:	NEW
Esempio:	NEW

Questo comando cancella il programma eventualmente contenuto nella memoria RAM e va usato quindi con molta cautela. Prima di scrivere NEW, dovete decidere se volete

conservare o no il programma attualmente in RAM; se lo volete conservare (per riprenderlo in qualche altra occasione), è necessario che il comando NEW sia preceduto da uno dei comandi SAVE o CSAVE (per conservare il programma su disco o, rispettivamente, su cassetta) seguito dal nome del programma stesso. L'istruzione NEW cancella anche la tabella dei simboli costruita dal BASIC. In questo modo tutte le variabili, i vettori e le stringhe definiti durante la conversazione precedente perdono il loro valore (vedi capitoli 7 ed 8). Il comando NEW può essere usato solo in modo diretto.

REM (R.)
[SPACE]

Formato: REM testo
Esempio: 10 REM ROUTINE PER CALCOLARE X

Questa istruzione, con tutto il testo che la segue sulla stessa riga logica, viene completamente ignorata dall'elaboratore durante l'esecuzione del programma. Essa dunque serve soltanto al programmatore, per scrivere dei commenti o per ricordare, in un certo punto del programma, qualsiasi informazione rilevante per lui. L'elaboratore, comunque, quando esegue il comando LIST, legge e riporta sullo schermo o sul nastro l'istruzione REM con tutto il testo che segue, come qualsiasi altra istruzione. Occorre però fare attenzione a non mettere altre istruzioni sulla stessa riga di un comando REM perché queste verrebbero considerate come un qualsiasi altro commento, e quindi ignorate.

RUN (RU.)

Formato: RUN [filespec]
Esempi: RUN
RUN "D:MENU"

Il comando RUN, che viene abitualmente dato in modo diretto, avvia l'esecuzione di un programma. Se, accanto a RUN, è riportata una filespec, l'elaboratore riprende, in forma codificata, il programma specificato dal file specificato, lo porta nella memoria RAM (sostituendolo ad un eventuale altro programma ivi residente) e lo esegue. Se il comando non è seguito da alcuna filespec, inizierà l'esecuzione del programma attualmente residente in RAM.

Il comando RUN assegna valore zero a tutte le variabili e chiude tutti i files ed i dispositivi periferici eventualmente aperti. Cancella dalla memoria tutti i vettori, le stringhe e le matrici precedentemente definiti ed interrompe tutti i suoni eventualmente attivi. Se durante l'esecuzione di un programma il BASIC incontra un errore, sullo schermo appare il messaggio relativo e l'esecuzione si interrompe (a meno che non sia stata usata l'istruzione TRAP, che serve a seguire il flusso del programma, "intrappolando gli errori").

Il comando RUN può essere usato anche in modo differito:

Esempi:

```
10 PRINT "OVER AND OVER AGAIN"
20 RUN
```

Se volete utilizzare il comando in modo diretto, scrivete RUN e premete **[RETURN].** Per interrompere l'esecuzione del programma premete **[BREAK]**.

L'esecuzione di un programma può anche iniziare da una riga numerata diversa dalla prima; per far ciò, non si usa il comando RUN ma il comando GOTO seguito dal numero della riga dalla quale si vuole iniziare e, come al solito, dal tasto **[RETURN]**.

STOP (STO.)

Formato: STOP
Esempio: 100 STOP

Quando il BASIC esegue un'istruzione STOP di un programma, scrive sullo schermo la scritta **STOPPED AT LINE** – – – – (interrotto alla riga – – – –), interrompe l'esecuzione del programma e ritorna al modo diretto. Il comando STOP non chiude i files e non interrompe i suoni eventualmente attivi. L'esecuzione del programma può quindi essere ripresa mediante il comando CONT **[RETURN]**.

CAPITOLO 3

FUNZIONI DI CONTROLLO DELLO SCHERMO (EDITING)

Oltre ai tasti che eseguono funzioni speciali (descritti nel capitolo 1), sulla tastiera dell'elaboratore ATARI ci sono dei tasti che svolgono funzioni di controllo del cursore e permettono di modificare rapidamente e semplicemente quello che appare sullo schermo. Per eseguire alcune di queste funzioni occorre premere insieme ai tasti indicati uno dei due tasti speciali di controllo che sono indicati con **[SHIFT]** e **[CTRL]**.

In questo capitolo descriveremo le funzioni di controllo dello schermo che sono regolate dai tasti o coppie di tasti seguenti:

[CTRL]	**[CTRL] [INSERT]**	**[CTRL] 1**
[SHIFT]	**[CTRL] [DELETE]**	**[CTRL] 2**
[CTRL] [↑]	**[SHIFT] [INSERT]**	**[CTRL] 3**
[CTRL] [↓]	**[SHIFT] [DELETE]**	**[BREAK]**
[CTRL] [→]	**[SHIFT] [CAPS/LOWR]**	**[ESC]**
[CTRL] [←]		

CONTROLLO DELLO SCHERMO

Tastiera e schermo sono collegati tra di loro: normalmente i caratteri che si battono sulla tastiera appaiono sullo schermo. Tuttavia quando si batte uno dei tasti di controllo sopra indicati, sullo schermo avvengono delle modifiche di tipo più complesso. L'insieme di queste possibilità di modifica si dice controllo dello schermo (Screen Editing). Il controllo dello schermo può servire anche per modificare i programmi. Se sullo schermo appare una riga di programma che contiene un errore, si può portare il cursore sopra l'errore e correggere sullo schermo l'errore stesso. Ma nella memoria del calcolatore l'immagine dello schermo è distinta da quella del programma, quindi la modifica apportata allo schermo deve essere riportata nel programma. A questo scopo occorre premere il tasto **[RETURN]** ogni volta che si è effettuata sullo schermo una correzione che deve essere riportata nel programma. Altrimenti la modifica rimane solo sullo schermo e sulla sua immagine in memoria, ma non altera il programma contenuto in una parte separata della stessa memoria.

Esempio:

```
10 REM PREMI RETURN DOPO LA COMPILAZIONE
DELLA RIGA
20 PRINT :PRINT
30 PRINT "QUESTA È LA RIGA 1 DELLO SCHERMO"
```

Per cancellare la riga 20 del programma, scrivere il numero di riga e premere **[RETURN]**. Se semplicemente cancellaste la riga dallo schermo, essa non verrebbe concellata dal programma.

Lo schermo e la tastiera, visti come dispositivi di I/O (Input/output, ingresso/uscita), verranno descritti nel capitolo 5.

[CTRL]
Tasto di controllo. Premendo questo tasto insieme ad uno dei tasti con le frecce si può modificare la posizione del cursore, spostarlo cioè in qualsiasi punto dello schermo, senza cancellare i caratteri che si verranno a trovare sotto al cursore stesso. **[CTRL]** inoltre, premuto assieme ad altri tasti, serve a controllare la definizione e la cancellazione dei valori di tabulazione, l'interruzione e la ripresa dell'elencazione di un programma, ed i simboli grafici particolari. Se, contemporaneamente al tasto **[CTRL]** si preme uno dei tasti che controllano tre funzioni, la funzione eseguita sarà quella indicata in alto a sinistra sul tasto stesso (vedi il capitolo 1).

[SHIFT]
Tasto delle maiuscole. Si può premere insieme ai tasti numerici per far apparire sullo schermo i simboli indicati sulla metà superiore di questi tasti. Oppure, se si preme insieme ad altri tasti, può servire a inserire o cancellare righe, per ritornare a scrivere in maiuscolo, o per scrivere i simboli funzionali che sono indicati sulla metà superiore dei tasti della sottrazione, dell'eguaglianza e della moltiplicazione; serve infine per scrivere le parentesi quadre, [ ], e il punto interrogativo, ? (vedi capitolo 1).

Tasti di controllo del cursore

FUNZIONI CONTROLLATE DA DUE TASTI

[CTRL] [↑]
Sposta il cursore sulla riga fisica superiore, senza disurbare il programma che è in RAM né modificare in alcun modo quello che appare sullo schermo.

[CTRL] [→]
Sposta il cursore di uno spazio a destra senza disurbare il programma o quanto appare sullo schermo.

[CTRL] [↓]
Sposta il cursore sulla riga fisica inferiore senza cambiare il programma o quanto appare sullo schermo.

[CTRL] [←]
Sposta il cursore di uno spazio a sinistra senza disturbare il programma o quanto appare sullo schermo.

Come gli altri tasti della tastiera ATARI, i tasti di controllo del cursore, se premuti più di mezzo secondo, ripetono l'operazione cui sono adibiti.

Tasti usati con [CTRL]

[CTRL] [INSERT]
Inserisce lo spazio per un carattere alla sinistra del cursore.

[CTRL] [DELETE]
Cancella lo spazio o il carattere alla sinistra del cursore.

[CTRL] 1
Sospende temporaneamente qualsiasi operazione senza uscire dal programma. Per riprendere le operazioni si preme di nuovo **[CTRL]** assieme al tasto **1**.

[CTRL] 2
Emette un segnale acustico.

[CTRL] 3
Indica la fine del file.

Tasti usati con [SHIFT]

[SHIFT] [INSERT]
Inserisce una riga logica.

[SHIFT] [DELETE]
Cancella una riga logica.

[SHIFT] [CAPS/LOWR]
Riporta alla scrittura in maiuscolo.

Tasti di controllo di funzioni speciali.

[BREAK]
Interrompe l'esecuzione o l'elencazione di un programma, sullo schermo scrive READY e fa apparire il cursore.

[ESC]
Fa sì che comandi generalmente usati in modo diretto possano essere usati in modo differito; per esempio, in modo diretto, **[CTRL] [CLEAR]** cancella lo schermo. Per cancellare lo schermo in modo differito, nel corso della stesura di un programma potete scrivere un numero di riga e poi, dopo la parola chiave PRINT e le virgolette, premere **[ESC]**, seguito da **[CTRL]** e **[CLEAR]** insieme:
PRINT "**[ESC] [CTRL] [CLEAR]**"

ISTRUZIONI PER IL CONTROLLO DEL FLUSSO NEI PROGRAMMI

In questo capitolo illustreremo le istruzioni che consentono di realizzare i cicli (loops), i salti condizionati e incondizionati, la ricerca di errori, l'uso di subroutines ed il loro richiamo. Spiegheremo anche i metodi che permettono l'accesso ai dati ed il comando opzionale per la definizione delle variabili.

Le istruzioni che descriveremo sono le seguenti:

FOR, TO, STEP/NEXT	**IF/THEN**	**POP**
GOSUB/RETURN	**ON, GOSUB**	**RESTORE**
GOTO	**ON, GOTO**	**TRAP**

FOR (F.), TO, STEP/NEXT (N.)

Formato: FOR avar = aexp1 to aexp2 (STEP aexp3)
NEXT avar

Esempi:

```
10 FOR X = 1 TO 10
100 NEXT X
10 FOR Y = 10 TO 20 STEP 2
100 NEXT Y
10 FOR INDEX = Z TO 100 * Z
100 NEXT INDEX
```

Incontrando queste istruzioni l'esecuzione del programma entra in un ciclo: le istruzioni che, nella lista del programma, sono comprese tra le istruzioni FOR... TO... e NEXT... verranno eseguite varie volte ed il numero di "passaggi" nel ciclo così instaurato è determinato anch'esso dalle istruzioni FOR/NEXT. La variabile indicata sopra con avar è detta variabile di ciclo. Ad essa viene inizialmente assegnato il valore indicato dall'aexp1. Ogni volta che, durante l'esecuzione del programma, si incontra l'istruzione NEXT avar, il valore assegnato alla variabile di ciclo aumenta della quantità indicata dall'aexp3 che segue STEP. Il valore di aexp3 può essere un numero intero positivo o negativo, un numero decimale o frazionario. Se l'istruzione STEP è assente, la variabile di ciclo viene incrementata di uno. Quando la variabile di ciclo supera il limite definito dall'aexp2, il ciclo si arresta, e l'esecuzione del programma riprende dall'istruzione immediatamente successiva all'istruzione NEXT e che può trovarsi sulla sua stessa riga o su quella successiva.

I cicli possono essere inseriti (nested) l'uno dentro l'altro. Quando ciò accade, viene eseguito per primo il ciclo più interno (che non ne contiene nessun altro). Una volta completato questo, si passa a quello immediatamente più esterno, e così via, sino all'esecuzione del ciclo più esterno di tutti.

Il programma seguente fornisce un esempio di due cicli inseriti l'uno nell'altro.

```
10 Z=0
20 FOR X=1 TO 3
30 PRINT "CICLO ESTERNO"
40 Z=Z+1
50 FOR Y=1 TO 5 STEP Z
60 PRINT "       CICLO INTERNO"
70 NEXT Y
80 NEXT X
90 END
```

Figura 4-1. Programma contenente due cicli inseriti l'uno nell'altro.

Nel programma della figura 4-1, il ciclo più esterno deve essere eseguito tre volte (con la variabile di ciclo, X, che va da 1 a 3 con incrementi di 1: ciò è sottinteso dall'assenza dell'istruzione STEP). Ma prima che il ciclo venga completato per la prima volta, ossia prima di giungere all'istruzione NEXT X, il programma passa il controllo al ciclo più interno, quello contraddistinto dalla variabile Y. La grammatica del BASIC vuole che l'istruzione NEXT Y relativa al ciclo più interno preceda quella (NEXT X) relativa al ciclo più esterno. Nel nostro caso il numero di esecuzioni del ciclo interno è determinato dall'istruzione STEP (STEP Z). Alla variabile Z è stato assegnato inizialmente il valore 0, prima dell'istruzione che darà il via all'esecuzione del ciclo esterno; poi, ad ogni passaggio per il ciclo esterno, il valore di Z aumenta di un'unità (Z=Z+1). Con la prima esecuzione del ciclo esterno il valore di Z sarà dunque 1, con la seconda 2, con la terza 3. Quando Z vale 1 il ciclo interno viene percorso cinque volte (con Y che assumerà i valori 1, 2, 3, 4 e 5). Al secondo passaggio per il ciclo esterno, quando Z vale 2, il ciclo interno viene percorso 3 volte (con Y che assume i valori 1, 3 e 5). Al terzo passaggio per il ciclo esterno, quando Z è 3, il ciclo interno viene percorso due sole volte (con Y che assume i valori 1 e 4).

L'esecuzione del programma è illustrata nella figura seguente.

```
CICLO ESTERNO
      CICLO INTERNO
      CICLO INTERNO
      CICLO INTERNO
      CICLO INTERNO
      CICLO INTERNO
CICLO ESTERNO
      CICLO INTERNO
      CICLO INTERNO
      CICLO INTERNO
CICLO ESTERNO
      CICLO INTERNO
      CICLO INTERNO
```

Figura 4-2. Esecuzione di cicli inseriti l'uno nell'altro.

Gli indirizzi di ritorno da un ciclo vengono collocati in uno speciale gruppo di indirizzi di memoria, detto "stack" (pila).

L'indirizzo viene mandato nello stack e, quando serve, ne viene estratto (vedi **POP)**

GOSUB (GOS.) RETURN (RET.)

Formato: GOSUB lineno
lineno
RETURN

Esempio: 100 GOSUB 2000
2000 PRINT "SUBROUTINE"
2010 RETURN

Una subroutine, generalmente, può fare qualsiasi cosa possa fare un programma. Viene utilizzata per risparmiare spazio di memoria o semplicemente per rendere i programmi più facili da leggere e da correggere.

Una subroutine è un programma o una routine relativamente autonoma che fa però parte di un programma più ampio ed è utilizzata, ad esempio, per calcolare qualche valore che poi ritorna al programma principale per essere utilizzato nell'esecuzione di un'istruzione successiva. Generalmente si ricorre ad una subroutine quando una stessa operazione deve essere ripetuta parecchie volte, all'interno di una sequenza di programma, usando gli stessi valori o valori differenti. L'istruzione GOSUB, contenuta nel programma principale, permette di "chiamare" la subroutine. L'ultima riga della subroutine deve contenere l'istruzione RETURN, che passa il controllo del programma alla riga immediatamente successiva all'istruzione GOSUB.

Come l'istruzione FOR/NEXT, anche GOSUB/RETURN usa uno stack per i suoi indirizzi di ritorno. Se ad una subroutine non viene permesso di completare normalmente le sue operazioni (se c'è, per esempio, un'istruzione GOTO su una riga che precede RETURN), l'indirizzo di ritorno dalla subroutine deve essere tolto dallo stack (vedi **POP**), altrimenti, in seguito, potrebbero verificarsi degli errori.

Poiché generalmente nel listato di un programma le istruzioni della subroutine seguono, nell'-ordine delle righe, il programma principale, potrebbe accadere che, terminata l'esecuzione del programma, scatti accidentalmente un nuovo avvio della subroutine semplicemente perché il BASIC continua a seguire la numerazione delle righe. Per prevenire ciò, è necessario porre un'istruzione END alla fine del programma, che preceda le righe della subroutine. Il seguente programma illustra l'uso delle subroutines.

```
10 PRINT "[ESC] [CLEAR]"
20 REM ESEMPIO DELL'USO DI GOSUB/RETURN
30 X=100
40 GOSUB 1000
50 X=120
60 GOSUB 1000
```

```
70 X=50
80 GOSUB 1000
90 END
1000 Y=3*X
1010 X=X+Y
1020 PRINT X, Y
1030 RETURN
```

Figura 4-3. Listato del programma GOSUB/RETURN.

In questo programma, la subroutine, che inizia alla riga 1000, viene chiamata tre volte, per calcolare e scrivere differenti valori di X e Y. La figura 4-4 illustra il risultato dell'esecuzione del programma.

```
400        300
480        360
200        150
```

Figura 4-4. Risultato dell'esecuzione del programma GOSUB/RETURN.

GOTO (G.)

Formato: GO TO aexp
GOTO

Esempi: 100 GOTO 50
500 GOTO (X + Y).

L'istruzione GOTO è un'istruzione di salto incondizionato, come l'istruzione GOSUB. Entrambe trasferiscono immediatamente il controllo del programma ad un altro numero di riga, indicato da una costante o da una espressione arbitraria. È comunque preferibile utilizzare una costante, altrimenti, nel caso in cui si vogliano rinumerare le righe del programma, il procedimento risulterebbe più complicato. Se un'istruzione GOTO indirizza verso un numero di riga che non esiste, verrà segnalato un errore. Un GOTO che salti ad una riga precedente può dar luogo ad un ciclo senza fine; infatti non verrebbero mai eseguite le istruzioni che seguono il GOTO. Per spezzare un ciclo GOTO si può usare un'istruzione del salto condizionato (vedi **IF/THEN**). Il seguente programma illustra due usi del comando GOTO.

```
10 PRINT
20 PRINT :PRINT "UNO"
30 PRINT "DUE"
40 PRINT "TRE"
50 PRINT "QUATTRO"
60 PRINT "CINQUE"
65 GOTO 100
70 PRINT "$$$$$$$$$$$$$$$$$"
```

```
 80 PRINT "%%%%%%%%%"
 90 PRINT "?????????????????"
 95 END
100 PRINT "SEI"
110 PRINT "SETTE"
120 PRINT "OTTO"
130 PRINT "NOVE"
140 PRINT "DIECI"
150 GOTO 70
```

Figura 4-5. Listato del programma GOTO.

Nel corso dell'esecuzione di questo programma, verranno scritti prima i numeri, poi le tre righe di simboli. I simboli delle righe 70, 80 e 90 vengono temporaneamente ignorati, mentre il programma esegue il comando GOTO 100. Poi vengono stampati i numeri dei "SEI" a "DIECI" ed infine viene eseguita la seconda istruzione GOTO che trasferisce il controllo del programma alla linea 70. (Naturalmente, questo è soltanto un esempio; il programma della figura 4-5 potrebbe essere riscritto senza far ricorso ad alcuna istruzione GOTO). L'esecuzione del programma produce il seguente risultato:

```
UNO
DUE
TRE
QUATTRO
CINQUE
SEI
SETTE
OTTO
NOVE
DIECI
$$$$$$$$$$$$$$$$$
%%%%%%%%%
?????????????????
```

Figura 4-6. Esecuzione del Programma GOTO.

IF/THEN

Formato: IF aexp THEN {lineno | statement [:statement...]}

Esempi:

```
IF X = 100 THEN 150
IF A$ = "ATARI" THEN 200
IF AA = 145 AND BB ≠ 1 THEN PRINT AA, BB
IF X = 100 THEN X=0
```

L'istruzione IF/THEN (se allora) è un'istruzione di salto condizionato. Il salto avviene solo se sono soddisfatte alcune condizioni (espresse dall'aexp che segue IF), che possono essere

aritmetiche o logiche. Se l'espressione che segue immediatamente IF è vera (cioè se ha valore diverso da zero), allora il programma esegue la parte THEN dell'istruzione. Se invece l'aexp è falsa (se cioè ha valore logico 0), il resto dell'istruzione viene ignorata ed il controllo del programma passa alla riga successiva.

Nel formato IF aexp THEN lineno, lineno deve essere una costante e non un'espressione, ed indica il numero della riga a cui dovrà passare il controllo del programma se la condizione posta da IF è vera. THEN può essere seguito da diverse istruzioni, separate dai due punti (:); in tal caso esse verranno eseguite tutte, ma, naturalmente, se e soltanto se la condizione IF è vera. Inoltre, nella stessa riga possono occorrere diverse condizioni IF, inserite una nell'altra. Per esempio:

```
100 IF X = 5 THEN IF Y = 3 THEN R = 9 : GOTO 200
```

L'istruzione R=9: GOTO 200 sarà eseguita se solo se X=5, e Y=3. Inoltre, la verità della condizione Y=3 verrà controllata se e solo se X=5.

Il seguente programma illustra l'uso dell'istruzione IF/THEN (ricordiamo che PRINT può essere abbreviato da ?).

```
5 GRAPHICS 0:? :? "           IF/THEN: DIMOSTRAZIONE"
10 ? :? "SCRIVI A"; :INPUT A
20 IF A=1 THEN 40 : REM QUALSIASI ISTRUZIONE POSTA
QUI NON VERREBBE MAI ESEGUITA!!
30 ? :? "A NON È 1. L'ESECUZIONE CONTINUA QUI QUANDO
L'ESPRESSIONE È FALSA".
40 IF A=1 THEN ? :? "A=1" :? "SI, È DAVVERO 1. ": REM QUESTE
DUE ISTRUZIONI VERRANNO ESEGUITE SOLO SE A=1!!
50 ? :? "L'ESECUZIONE CONTINUA QUI SE A<>1 OPPURE DOPO
CHE È APPARSA LA SCRITTA SI, È DAVVERO 1".
60 GOTO 10
```

Figura 4-7. Listato di programma con IF/THEN.

```
        IF/THEN: DIMOSTRAZIONE
SCRIVI A                                    (l'utente scrive 2)

A NON È 1. L'ESECUZIONE CONTINUA QUI QUANDO
L'ESPRESSIONE È FALSA.
L'ESECUZIONE CONTINUA QUI SE A<>1 OPPURE
DOPO CHE È APPARSA LA SCRITTA "SI, È DAVVERO 1"

SCRIVI A                                    (l'utente scrive 1)

A=1
SI, È DAVVERO 1.
L'ESECUZIONE CONTINUA QUI SE A<>1 OPPURE
DOPO CHE È APPARSA LA SCRITTA "SI, È DAVVERO 1"
```

SCRIVI A

Figura 4-8. Esecuzione del programma IF/THEN.

ON/GOSUB/ RETURN ON/GOTO

Formato: ON aexp { GOTO | GOSUB } lineno [,lineno...]

Esempi:

```
100 ON X GOTO 200, 300, 400
100 ON A GOSUB 1000, 2000
100 ON SQR (X) GOTO 30, 10, 100
```

Nota: GOSUB e GOTO non possono essere abbreviati.

Anche queste due istruzioni sono istruzioni di salto condizionato, come IF/THEN. Sono però molto più potenti.

L'aexp deve avere come valore un numero positivo che viene arrotondato al più vicino valore intero positivo, fino a 255. Se il numero risultante è 1, allora il controllo del programma passa alla riga indicata dal primo numero che segue la parola chiave GOSUB o GOTO. Se il numero risultante è 2, il controllo del programma passa alla riga indicata dal secondo numero che segue GOSUB o GOTO, e così via. Se il valore dell'aexp è 0, oppure è maggiore del numero di linee che seguono la parola chiave GOSUB o GOTO, allora non sono soddisfatte le condizioni necessarie perché venga eseguita l'istruzione ON/GOSUB o ON/GOTO ed il controllo del programma passa all'istruzione successiva, che può trovarsi o non trovarsi sulla stessa riga. Con ON/GOSUB, il controllo del programma passa alla subroutine indicata e poi, come dopo l'esecuzione di una subrotine chiamata da un'istruzione GOSUB, ritorna alla riga successiva.

Il programma seguente illustra l'uso dell'istruzione ON/GOTO:

```
10 X=X+1
20 ON X GOTO 100, 200, 300, 400, 500
30 IF X>5 THEN PRINT "HO FINITO" END
40 GOTO 10
50 END
100 PRINT "ORA STO LAVORANDO ALLA RIGA 100" : GOTO 10
200 PRINT "ORA STO LAVORANDO ALLA RIGA 200" : GOTO 10
300 PRINT "ORA STO LAVORANDO ALLA RIGA 300" : GOTO 10
400 PRINT "ORA STO LAVORANDO ALLA RIGA 400" : GOTO 10
500 PRINT "ORA STO LAVORANDO ALLA RIGA 500" : GOTO 10
```

Figura 4-9. Listato del programma ON/GOTO.

L'esecuzione del programma darà il seguente risultato:

ORA STO LAVORANDO ALLA RIGA 100

```
ORA STO LAVORANDO ALLA RIGA 200
ORA STO LAVORANDO ALLA RIGA 300
ORA STO LAVORANDO ALLA RIGA 400
ORA STO LAVORANDO ALLA RIGA 500
HO FINITO
```

Figura 4-10. Esecuzione del programma ON/GOTO.

POP

Formato: POP
Esempio: 1000 POP

Nella descrizione di FOR/NEXT, abbiamo definito lo stack come un gruppo di indirizzi di memoria riservato ad indirizzi di ritorno. La prima informazione collocata nello stack riguarda il numero di cicli da eseguire o, nell'esecuzione di un GOSUB, il numero della riga di destinazione a cui rimanda il RETURN della subroutine. Se una subroutine non termina con un'istruzione RETURN, la locazione superiore della memoria conterrà ancora alcuni numeri. Quindi, se il programma prevede l'esecuzione di un altro GOSUB, sarà necessario cancellare prima i dati da tale locazione. Per preparare lo stack per un nuovo GOSUB, si usa l'istruzione POP, che toglie i dati dalla locazione superiore dello stack.

Nell'utilizzare un'istruzione POP, si devono osservare le regole seguenti:

1. L'istruzione deve essere collocata in modo che l'esecuzione del programma la incontri, e non, ad esempio, nella stessa subroutine, in una riga successiva all'istruzione GOTO che fa uscire in modo anomalo dalla subroutine.
2. Deve seguire la riga di un'istruzione GOSUB che passa il controllo ad una subroutine la quale, a sua volta, non termina con l'istruzione RETURN e dunque non riporta al programma principale.

Il seguente esempio illustra l'uso del comando POP associato a GOSUB, quando nella subroutine non verrà eseguita l'istruzione RETURN.

```
10 GOSUB 1000
15 REM LA RIGA 20 NON SARÀ ESEGUITA
20 PRINT "UN RITORNO NORMALE DÀ QUESTO MESSAGGIO"
30 PRINT "UN RITORNO ANORMALE DÀ QUESTO MESSAGGIO"
40 POP
999 END
1000 PRINT "STO ORA ESEGUENDO LA SUBROUTINE"
1010 GOTO 30
1020 RETURN
```

Figura 4-11. Istruzione GOSUB con POP.

RESTORE (RES.)

Formato: RESTORE [aexp]
Esempio: 100 RESTORE

Il sistema di elaborazione personale ATARI ha un contatore interno che contiene il numero di dati dell'istruzione DATA già letti e dunque tiene traccia dell'elemento che dovrà essere letto nell'eseguire la successiva istruzione READ. Utilizzata senza l'aexp opzionale, l'istruzione RESTORE riporta il contatore al primo elemento di DATA del programma. Se invece RESTORE è seguita da una aexp, il contatore viene riportato al primo elemento di quell'istruzione DATA che si trova sulla riga specificata dal valore dell'aexp. L'istruzione RESTORE permette dunque di usare più di una volta gli stessi dati.

```
10 FOR N=1 TO 2
20 READ A
30 RESTORE
40 READ B
50 M=A+B
60 PRINT "IL TOTALE DELLA SOMMA È ";M
70 NEXT N
80 END
90 DATA 30, 15
```

Figura 4-12. Listato del programma RESTORE.

Alla prima esecuzione del ciclo. A sarà 30 e B sarà 30 cosicché la riga del totale 50 scriverà IL TOTALE DELLA SOMMA È 60, ma alla seconda esecuzione del ciclo, A sarà uguale a 15 e B, per via dell'istruzione RESTORE, sarà ancora eguale a 30. Perciò, l'istruzione PRINT della riga 50 farà apparire IL TOTALE DELLA SOMMA È 45.

TRAP (T.)

Formato: TRAP aexp
Esempio: 100 TRAP

L'istruzione TRAP viene utilizzata per passare il controllo del programma alla riga specificata dall'aexp se, nel corso dell'esecuzione, si incontra un errore. Senza aver ricevuto un'istruzione TRAP, quando il BASIC incontra un errore interrompe l'esecuzione del programma e sullo schermo appare il relativo messaggio di errore.

L'istruzione TRAP "intrappola" qualsiasi errore possa occorrere dopo la sua esecuzione. Tuttavia, una volta che è stato scoperto un errore ed evitata così l'interruzione che esso avrebbe provocato, è necessario riposizionare la trappola, con un'altra istruzione TRAP; poiché generalmente gli errori provengono con i dati immessi dall'utente in una determinata esecuzione, è conveniente collocare l'istruzione TRAP all'inizio della sezione del programma che concerne gli ingressi provenienti dalla tastiera, in modo che essa venga riattivata dopo ogni errore. PEEK(195) vi darà il codice dell'errore commesso (vedi Appendice B), mentre 256*PEEK(187)+PEEK(186) vi darà il numero della riga in cui è stato scoperto

l'errore; così, se volete sapere quale errore è stato commesso e dove, potete dare i comandi PRINT PEEK (195) e PRINT 256*PEEK(187)+PEEK(186). Si può cancellare la trappola per gli errori dando un'istruzione in cui TRAP è seguito da una aexp il cui valore va da 32767 a 65535 (per esempio, 40000).

NOTE

DISPOSITIVI E COMANDI DI INPUT/OUTPUT

In questo capitolo descriveremo i dispositivi di input/output (ingresso/uscita) e come si possano trasferire i dati da uno di tali dispositivi ad un altro. I comandi di cui parleremo sono cioè quelli che permettono di accedere ai dispositivi di input/output. I comandi di ingresso sono quelli che permettono di introdurre dati nella memoria RAM e nei dispositivi progettati per l'accettazione dell'input. I comandi di uscita sono quelli associati al recupero dei dati presenti in RAM e nei dispositivi progettati per la produzione di output.

Descriveremo i comandi seguenti:

CLOAD	**INPUT**	**OPEN/CLOSE**	**READ/DATA**
CSAVE	**LOAD**	**POINT**	**SAVE**
DOS	**LPRINT**	**PRINT**	**STATUS**
ENTER	**NOTE**	**PUT/GET**	**XIO**

DISPOSITIVI DI INPUT/OUTPUT

La configurazione dell'"hardware' dei dispositivi di cui parleremo è descritta in dettaglio negli specifici manuali ad essi allègati. Il sottosistema centrale di input/output (CIO = Central Input/Output) fornisce all'utente un'unica interfaccia per accedere a tutti i dispositivi periferici del sistema in modo largamente indipendente. Ciò significa che c'è un unico punto di entrata ed una sequenza chiamante indipendente dal dispositivo. Ogni dispositivo ha uno specifico nome simbolico, che serve per identificarlo; **K:**, per esempio, è il nome della tastiera (Keyboard). Ognuno di questi dispositivi, prima dell'accesso, deve essere aperto ed assegnato ad un blocco di controllo dell'input/output (IOCB = Input/Output Control Block). Da quel momento in poi il dispositivo verrà denotato dal suo numero di IOCB.

Il BASIC ATARI contiene, in RAM, 8 blocchi di controllo dell'input/output destinati a contenere le informazioni di cui ha bisogno il sistema operativo per eseguire un'operazione I/O (di input/output). Queste informazioni comprendono il comando, la lunghezza della memoria di transito (buffer) e il suo indirizzo, e due variabili ausiliarie di controllo. Il BASIC ATARI prepara i blocchi di controllo dell'input/output, ma l'utente deve specificare quale IOCB vuole utilizzare. Il BASIC riserva il blocco di controllo dell'input/output numero 0 (l'IOCB#0) per le operazioni riguardanti il controllo dello schermo; l'utente perciò non può richiedere, per aprire un altro dispositivo, l'IOCB#0. L'istruzione GRAPHICS (vedi Capitolo 9) apre l'IOCB#6 per l'ingresso e l'uscita da e per lo schermo (cioè per la finestra della grafica, denotata da **S:**). L'IOCB#7 viene utilizzato dal BASIC per i comandi LPRINT, CLOAD e CSAVE. Ci si può anche riferire al numero di IOCB come al numero di dispositivo (o di file). Gli IOCB numerati da 1 a 5 vengono utilizzati per aprire gli

altri dispositivi destinati ad eseguire operazioni di input/output. Se si sta utilizzando l'IOCB#7, non sarà possibile eseguire l'istruzione LPRINT ed alcune delle altre istruzioni I/O del BASIC.

Tastiera: (K:) Dispositivo adibito soltanto alle operazioni in input. Quando preme un tasto, l'utente «scrive» il dato in cui viene convertito l'input inviato dal tasto stesso (codice ATASCII del carattere).

Stampante (P:) Dispositivo adibito soltanto alle operazioni di output. Stampa i caratteri ATASCII, una riga per volta. Non riconosce i caratteri di controllo.

Registratore di programmi (C:) Dispositivo di ingresso e uscita. Il registratore funziona come dispositivo per la lettura o la scrittura (registrazione) di dati, ma, ovviamente, non può compiere queste due azioni simultaneamente. La cassetta ha due canali: una traccia serve per la registrazione di dati e programmi, e l'altra, la traccia audio, consente di registrare suoni e voci. La traccia audio non può essere incisa direttamente dal sistema ATARI, ma i suoni registrati si possono ascoltare utilizzando l'altoparlante del televisore.

Unità disco (Disk Drives): (D1:, D2:, D3:, D4:) Dispositivi di input e di output. Richiedono un minimo di 16 K di Ram. L'elaboratore personale ATARI può controllare fino a quattro unità disco. Se è collegato solo ad una unità disco, non è necessario aggiungere alcun numero dopo il nome simbolico del dispositivo, **D.**

Dispositivo di controllo dello schermo (Screen Editor): (E:) Dispositivo di input e di output. E' composto da schermo e tastiera (vedi Monitor TV). Per mezzo della tastiera l'utente può inviare informazioni al dispositivo ed i dati appariranno sullo schermo, a partire dalla posizione in cui si trova il cursore. Le informazioni che invece l'utente riceve, in forma di scritte sullo schermo, gli permettono di controllare lo schermo ed i dati introdotti. Quando si preme il tasto **[RETURN]**, l'intera riga logica su cui si trova il cursore viene individuata come "record" (un record è un insieme di informazioni considerate come un blocco unico) che il sistema CIO trasferirà nel programma dell'utente (vedi capitolo 9).

Monitor TV (S:): Dispositivo di ingresso e uscita. Permette all'utente di leggere i caratteri e di scriverli sul video, utilizzando il cursore per orientarsi sullo schermo stesso. Potete scrivere (effettuare operazioni di "testo") oppure eseguire operazioni di tipo grafico. Nel capitolo 9 descriveremo in dettaglio le modalità grafiche.

Interfaccia RS-232 (R:): Il dispositivo RS-232 permette al sistema ATARI di interfacciare con i dispositivi compatibili con l'RS-232, come stampanti, terminali e tracciatori di curve (plotters). Possiede una porta parallera a cui può essere connessa la stampante a 80 colonne (ATARI 825™).

CLOAD (CLOA.)

Formato: CLOAD
Esempi: CLOAD
100 CLOAD

Questo comando si può usare sia in modo diretto che in modo differito e serve per caricare in RAM, da una cassetta, un programma da eseguire. Quando si dà il comando CLOAD, il

suono di un campanello segnala che bisogna premere il tasto PLAY del registratore, seguito da **[RETURN]**. Non premere, comunque il tasto PLAY finché il nastro non sia stato collocato nel vano-cassetta. Istruzioni specifiche per il caricamento di un programma tramite CLOAD, sono contenute nell'"ATARI 410 Program Recorder Manual" (Manuale per l'uso del registratore di programmi ATARI 410). Alla fine del capitolo, nel paragrafo **CONCATENAZIONE DI PROGRAMMI**, descriveremo il procedimento da seguire per caricare programmi che superano una certa lunghezza.

CSAVE (CS.)

Formato: CSAVE
Esempi: CSAVE
100 CSAVE
100 CS.

Questo comando viene utilizzato generalmente in modo diretto, per registrare su nastro un programma residente in RAM. CSAVE conserva il programma nella sua versione codificata. Quando si dà il comando CSAVE, due suoni di campanello indicano che bisogna premere i tasti PLAY e RECORD (del registratore), seguiti da **[RETURN]** (sulla tastiera). Non bisogna, comunque, premere tali tasti, prima di aver ben inserito la cassetta nel registratore. È più rapido salvare un programma usando questo comando piuttosto che SAVE "C" (vedi **SAVE**) poiché con CSAVE gli intervalli tra i records sono molto più brevi.

Note:
Due nastri sui quali sia stato registrato utilizzando i due diversi comandi SAVE e CSAVE non sono compatibili.

Prima di ogni comando CSAVE si dovrebbe dare il comando LPRINT (vedi LPRINT), altrimenti CSAVE potrebbe funzionare male.

Per istruzioni specifiche sulle modalità di connessione e di avviamento dell'hardware, delle cassette, etc., consultare l'"ATARI 410 Program Recorder Manual".

DOS (DO.)

Formato: DOS
Esempio: DOS

Questo comando si usa per passare dal BASIC al sistema operativo su disco (DOS). Se il sistema operativo su disco non è stato messo in memoria, l'elaboratore andrà a finire nel modo di tacquino d'appunti (Memo Pad Mode). Per tornare al modo diretto, l'utente dovrà allora premere **[RESET]**. Quando il sistema operativo su disco è stato caricato in memoria, sul video apparirà il Menu DOS (la lista delle possibilità che il DOS propone). Per cancellare dallo schermo il Menu DOS, premere **[RESET]**. Il controllo tornerà allora al BASIC. Potete tornare al BASIC anche scegliendo dal Menu DOS la lettera di comando **B** (Caricare la cartuccia).

Il comando DOS si usa generalmente in modo diretto, ma si può anche utilizzare in un programma. Per ulteriori dettagli, consultare l'"ATARI DOS Manual" (Manuale del DOS Atari).

ENTER (E.)

Formato: ENTER filespec
Esempi: ENTER "C
ENTER "D:DEMOPR.INS"

Questo comando si usa per trasferire in memoria, dal nastro o dal disco, un programma registrato originariamente usando LIST (vedi **LIST** nel capitolo 2). Il programma è immesso in forma non codificata e il BASIC, quando riceve i dati, li interpreta. Quando il caricamento è completo, il programma può essere eseguito nel modo solito. È da notare che sia LOAD che CLOAD (vedi capitolo 2) cancellano il vecchio programma dalla memoria prima di caricare il nuovo; ENTER invece fonde il programma vecchio con quello nuovo. ENTER si usa generalmente in modo diretto.

INPUT (I.)

Formato: INPUT [#aexp {,|;}] {avar|svar} [, {avar|svar} ...]
Esempi: 100 INPUT X
100 INPUT N$
100 PRINT "IMMETTI IL VALORE X"
110 INPUT X

Eseguendo questa istruzione, l'elaboratore attende che l'utente immetta, utilizzando la tastiera, uno o più dati. Nel corso dell'esecuzione, quando il BASIC incontra un'istruzione INPUT, appare sullo schermo un punto interrogativo (?), indicando così che è pronto a ricevere i dati. Un'istruzione INPUT viene generalmente preceduta da un'istruzione PRINT che informi l'utente sul tipo di dati che l'elaboratore gli richiede.

La parola chiave INPUT può essere seguita anche da una variabile stringa, purché non sia indicizzata. Non sono ammesse invece le variabili matrice.

L'#aexp è opzionale e serve a specificare il numero di file o di dispositivo da cui dovranno provenire i dati (vedi Dispositivi di input/output). Se non è specificata alcuna #aexp, allora il BASIC aspetterà un input proveniente dalla tastiera (è sottinteso, cioè, che il dispositivo emittente sarà il dispositivo di controllo dello schermo, **E:**).

Se in un'istruzione la parola chiave INPUT è seguita da più di una avar ed i dati devono provenire dal dispositivo di controllo dello schermo (ad esempio, INPUT X, Y, Z), al momento dell'esecuzione l'utente può scrivere i numeri corrispondenti tutti sulla stessa riga, purché siano separati da una virgola, oppure su righe distinte. Nel caso in cui invece un'istruzione INPUT richieda l'immissione di diverse stringhe, quando appare il punto interrogativo, durante l'esecuzione, l'utente deve immettere la prima stringa e premere **[RETURN]**; sulla riga successiva apparirà un altro punto interrogativo e l'utente può allora scrivere la seconda stringa, premere **[RETURN]**, etc. È evidente che debba essere così se si pensa che due stringhe separate da una virgola apparirebbero al BASIC come una stringa unica.

```
10 PRINT "SCRIVI 5 NUMERI DA SOMMARE"
20 FOR N=1 TO 5
30 INPUT X
40 C=C+X
50 NEXT N
60 PRINT "LA SOMMA DEI NUMERI DATI È ";C
```

Figura 5. Listato di un programma con INPUT.

La figura illustra il programma per l'addizione di 5 numeri scelti dall'utente.

LOAD (LO.)

Formato: LOAD filespec
Esempio: LOAD "D1:SUSANNA.AGN"

Questo comando è simile a CLOAD tranne per il fatto che la parola chiave deve essere seguita dalla specificazione del file, e può quindi servire per caricare sia dal registratore che dalle unità disco. LOAD inserisce tra i records spazi piuttosto lunghi (vedi invece **CLOAD**) ed una versione codificata del programma. Quando si usa una sola unità disco, non è necessario specificare alcun numero dopo la "D", perché in tal caso si sottintende l'uso dell'unità disco numero 1.

LPRINT (LP.)

Formato: LPRINT [exp] [{ ; , } exp...]

Esempi: LPRINT "PROGRAMMA PER CALCOLARE X"
100 LPRINT X;" ";X;" ";Z

Quando riceve questa istruzione, l'elaboratore stampa i dati sulla stampante anziché sullo schermo. Può essere usata in modo diretto o in modo differito. Non richiede la specificazione del dispositivo né istruzioni OPEN e CLOSE. (Il BASIC usa l'IOCB#7).

Per stampare il listato di un programma sulla stampante vedi **LIST.**

NOTE (NO)

Formato: NOTE #aexp, avar, avar
Esempio: 100 NOTE #1, X, Y

Questo comando serve per assegnare l'attuale numero di settore del disco alla prima variabile aritmetica (avar) e l'attuale numero di byte all'interno del settore, alla seconda avar. Questa è la posizione di lettura o scrittura, nel file specificato, del byte da leggere o scrivere successivamente. Il comando NOTE si usa quando si scrivono dati su un file di disco (vedi **POINT**). L'informazione data dal comando NOTE viene scritta in un secondo file che si usa poi come indice del primo file.

OPEN (O.) CLOSE (CL.)

Formato: OPEN #aexp, aexp1, aexp2, filespec
CLOSE #aexp

Esempi:
100 OPEN #2,8,0,"D1:ATARI800.BAS"
100 A$ = "D1:ATARI800.BAS"
110 OPEN #2,8,0,A$
150 CLOSE #2

Per poter accedere ad un dispositivo, bisogna "aprirlo". Il processo di "apertura" serve a legare uno specifico IOCB al gestore del dispositivo in questione, inizializza le variabili di controllo collegate al CIO, e passa ogni opzione specifica del dispositivo al gestore del dispositivo stesso. I parametri per il comando OPEN sono definiti come segue:

— Carattere obbligatorio che deve essere immesso dall'utente.

aexp — Numero di IOCB o di file di riferimento per ogni utilizzazione successiva (come nel comando CLOSE). Il numero può andare da 1 a 7.

aexp 1 — Numero di codice per determinare quali operazioni di input o output si eseguiranno.

Codice 4 = operazione di ingresso.

Codice 8 = operazione di uscita.

Codice 12 = operazioni di ingresso e uscita.

Codice 6 = operazione di accesso al direttorio del disco (in questo caso il filespec è la specificazione della ricerca).

Codice 9 = operazione (di uscita) che aggiunge il simbolo di fine del file. Questo simbolo è usato anche per realizzare una particolare modalità di input che permette ad un programma di accettare dal dispositivo di controllo dello schermo, E: la riga successiva senza aspettare che l'utente prema **[RETURN]**.

aexp2 — Codice ausiliario dipendente dal dispositivo. Il numero 83, come valore di questo parametro, indica i margini per la scrittura su stampante (vedi l'apposito manuale per i codici di controllo).

filespec — Specificazione del file. Il nome deve essere racchiuso tra virgolette. Il formato di questo parametro è illustrato nella figura 5-2.

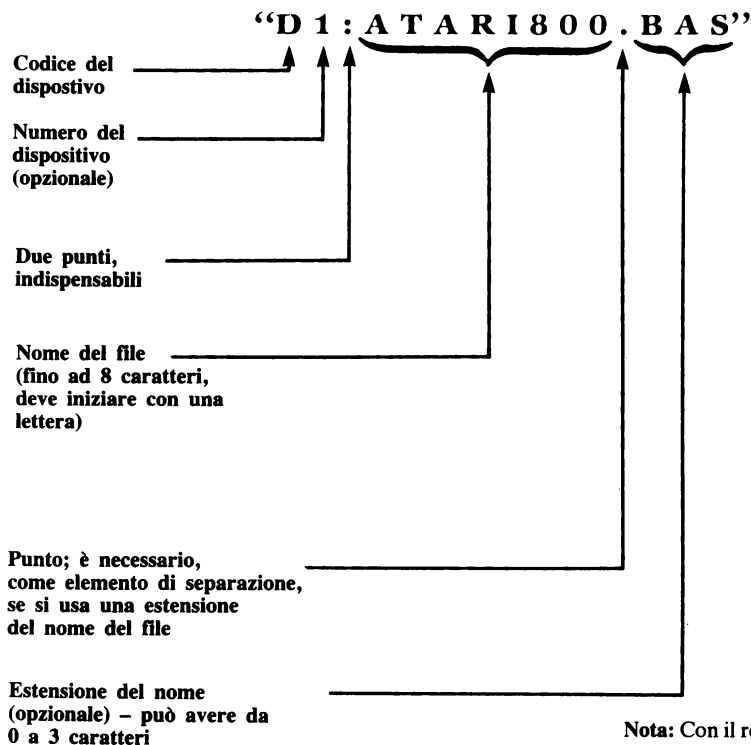



Figura 5-2. Struttura dei nomi di files.

Il comando CLOSE chiude semplicemente i files aperti precedentemente dal comando OPEN. Nell'esempio si deve notare che l'aexp che segue il carattere obbligatorio # deve essere eguale al numero di riferimento indicato dall'aexp nel comando OPEN.

POINT (P.)

Formato: POINT #aexp, avar, avar
Esempio: 100 POINT #2, A, B

Questo comando si usa per trasferire un file in RAM. La prima avar specifica il numero di settore del disco, la seconda il byte all'interno di quel settore in cui verrà letto o scritto il byte successivo. Questo comando, in pratica, sposta un puntatore controllato dal software verso una posizione specifica nel file. Ciò dà all'utente un accesso casuale ai dati conservati in un file di disco. I comandi POINT e NOTE sono discussi in maggior dettaglio nel Manuale del sistema operativo DOS.

PRINT (PR oppure?)

Formato: PRINT [#aexp] { ; , } [exp] [,exp...]

Esempi:
```
PRINT X, Y, Z, A$
100 PRINT "IL VALORE DI X È ";X
100 PRINT "VIRGOLE", "COLONNA", "SPAZIATURA"
100 PRINT #3, A$
```

Il comando PRINT si può usare sia in modo diretto che in modo differito. Nel modo diretto PRINT stampa qualsiasi informazione sia contenuta tra le virgolette, esattamente come essa appare. Nel primo esempio, PRINT X,Y,Z,A$, lo schermo mostrerà i valori che il programma residente in RAM ha assegnato alle variabili X,Y,Z e A$. Nell'ultimo esempio, #3 è la specificazione di file (può essere un numero da 1 a 7) che controlla su quale dispositivo verrà stampato il valore di A$ (vedere **Dispositivi di input/output**).

Se due espressioni che seguono PRINT sono separate da una virgola, la seconda sarà scritta a 10 spazi di distanza dal primo carattere dell'espressione precedente (se questa non è troppo lunga). Se invece esse sono separate da un punto e virgola, il BASIC scriverà la seconda espressione immediatamente dopo quella precedente, senza lasciare alcuno spazio tra di esse. Perciò nel secondo esempio, prima della chiusura delle virgolette, è stata posta una spaziatura, in modo che il valore di X non venga scritto immediatamente dopo la parola "È". Se alla fine di un comando PRINT non si mette né virgola né punto e virgola, allora, l'elaboratore eseguirà un ritorno carrello: il PRINT successivo partirà cioè dalla riga seguente.

PUT (PU.)/ GET (GE.)

Formato:
```
PUT #aexp, aexp
GET #aexp, avar
```

Esempi:
```
100 PUT #6, ASC("A")
200 GET #1,X
```

I comandi PUT e GET sono opposti tra loro. Il comando PUT invia un singolo byte, ossia un intero compreso tra 0 e 255, contenuto in aexp, al file specificato da #aexp (# è un carattere obbligatorio per entrambi questi comandi). Il comando GET legge un byte (usando #aexp per designare il file, su dischetto o altro), e poi assegna tale byte alla variabile avar (per alcune applicazioni di questi comandi, vedi il capitolo 9).

READ (REA)/ DATA (D.)

Formati:
```
READ var [, var...]
DATA adata [, adata...]
```

Esempi:
```
100 READ A,B,C,D,E
110 DATA 12,13,14,15,16
100 READ A$,B$,C$,D$,E$
110 DATA FULVIA, GIOVANNI, VITTORIO, LUCIANA,
ROSA
```

Questi due comandi vanno usati sempre insieme, e l'istruzione DATA si può usare soltanto in modo differito. (Un READ in modo diretto leggerà il dato solo se è stata già eseguita un'istruzione DATA di un programma). L'istruzione DATA può essere posta ovunque nel programma ma il numero di dati che essa contiene non deve essere minore del numero di dati che l'istruzione READ dovrà leggere. Altrimenti apparirà sullo schermo l'indicazione di errore "dati insufficienti".

Se la parola chiave READ è seguita da una o più variabili stringa, esse devono già essere state "dimensionate" e non possono essere variabili indicizzate (vedi il capitolo **STRINGHE**). Inoltre in un'itruzione READ non si possono usare variabili vettore.

L'istruzione DATA conserva, sotto forma di stringa, un certo numero di dati ai quali possa accedere l'istruzione READ. Non può includere operazioni aritmetiche, funzioni, etc. Inoltre il tipo di dati in DATA deve corrispondere al tipo di variabili occorrenti nella corrispondente istruzione READ.

Il programma seguente esegue la somma della lista di numeri contenuti nell'istruzione DATA:

```
10 FOR N=1 TO 5
20 READ D
30 M=M+D
40 NEXT D
50 PRINT "IL TOTALE DELLA SOMMA È ";M
60 END
70 DATA 30,15,106,17,87
```

Figura 5-3. Listato di un programma con READ/DATA.

Eseguendo il programma, il BASIC scriverà:

```
IL TOTALE DELLA SOMMA È 255
```

SAVE (S.)

Formato: SAVE filespec
Esempio: SAVE "D1:SANDRO.PER"

Il comando SAVE è simile a CSAVE tranne per il fatto che esso deve essere seguito dal nome del file e può quindi servire per salvare sia su disco che su nastro. Se si usa una sola unità disco, si può omettere il numero di codice di dispositivo: la mancanza di tale numero sottintende l'unità disco numero 1. SAVE, come LOAD, inserisce tra i records, su cassetta, spazi piuttosto lunghi (vedi **CSAVE**) e salva il programma nella sua forma non codificata.

STATUS (ST.)

Formato: STATUS #aexp, avar
Esempio: 350 STATUS #1,Z

Il comando STATUS chiama la routine STATUS del dispositivo specificato (aexp). Lo status del dispositivo (v. **CODICI DI ERRORE,** Appendice B), è immagazzinato nella variabile indicata (avar). Ciò può essere utile per altri dispositivi, come l'interfaccia RS-232.

XIO (X.)

Formato: XIO cmdno, #aexp, aexp1, aexp2, filespec
Esempio: XIO 18,#6,0,0,"S:"

Il comando XIO è un'istruzione di input/output che serve per eseguire operazioni particolari. Ad esempio, si può utilizzare per riempire con un colore un'area dello schermo compresa tra punti e linee precedentemente tracciati (vedi il capitolo 9). I parametri di XIO sono definiti come segue:

cmdno Numero che indica il particolare comando da eseguire

cmdno	OPERAZIONE	ESEMPIO
3	OPEN	Lo stesso che OPEN del BASIC
5	GET RECORD	Questi 4 comandi sono simili,
7	GET CHARACTERS	rispettivamente, ai comandi
9	PUT RECORD	del BASIC INPUT, GET, PRINT
11	PUT CHARACTERS	e PUT
12	CLOSE	Lo stesso che CLOSE del BASIC
13	STATUS REQUEST	Lo stesso che STATUS del BASIC
17	DRAW LINE	Lo stesso che DRAWTO del BASIC
18	FILL	Vedi capitolo 9
32	RENAME	XIO 32,#1,0,0,"D:TEMP.CAROL"
33	DELETE	XIO 33,#1,0,0,"D:TEMP.BAS"
35	LOCK FILE	XIO 35,#1,0,0,"D:TEMP.BAS"
36	UNLOCK FILE	XIO 36,#1,0,0,"D:TEMP.BAS"
37	POINT	Lo stesso che POINT del BASIC
38	NOTE	Lo stesso che NOTE del BASIC
254	FORMAT	XIO 254,#1,0,0,"D2:"

aexp Numero di dispositivo (come in OPEN). Nella maggior parte dei casi esso viene ignorato, ma deve comunque essere preceduto da #.

aexp1 e aexp2 Due bytes di controllo ausiliari. Il loro uso dipende dal dispositivo o dal comando particolari. Nella maggior parte dei casi non vengono usati e si pongono eguali a 0.

filespec Espressione stringa che specifica il dispositivo. Deve essere racchiusa tra virgolette. Sebbene alcuni comandi, come quello per l'operazione di riempimento (vedi il capitolo 9), non dipendano dalla filespec, tuttavia è obbligatorio includerla nell'istruzione.

CONCATENAZIONE DI PROGRAMMI

Se un programma richiede più memoria di quella disponibile, potete effettuare le seguenti operazioni ed unire assieme programmi di lunghezza accettabile.

1. Scrivere normalmente la prima parte del programma.
2. L'ultima riga della prima parte del programma conterrà soltanto il numero di riga e il comando RUN"C:"
3. Avvolgere la cassetta fino ad una parte libera del nastro. Segnarsi il numero del contatore che corrisponde all'inizio del programma per successivi comandi RUN. Premere i tasti PLAY e RECORD sul registratore in modo che entrambi rimangano abbassati.
4. Scrivere SAVE "C:" e premere **[RETURN]**
5. Quando si sente il suono di avvertimento, premere di nuovo **[RETURN]**.
6. Quando sullo schermo appare "READY", non rimuovere la cassetta. Scrivere NEW **[RETURN]**.
7. Ripetere le operazioni precedenti per la seconda parte del programma.
8. Poiché la seconda parte del programma è in sostanza un programma totalmente nuovo, è possibile riutilizzare gli stessi numeri di riga della prima parte del programma.
9. Se c'è una terza parte di programma assicuratevi che l'ultima riga della seconda parte sia un comando RUN"C:".

Per l'esecuzione di un programma "a catena" compiere le seguenti operazioni:

1. Riavvolgere il nastro fino all'inizio della parte 1 del programma.
2. Premere il tasto PLAY sul registratore.
3. Scrivere RUN"C:" **[RETURN]**.
4. Quando si sente il suono di avvertimento, premere di nuovo **[RETURN]**.

L'elaboratore carica automaticamente la prima parte del programma, la esegue ed emette un suono per indicare che è necessario premere il tasto lungo di spaziatura o **[RETURN]** per avviare il motore del registratore per il secondo LOAD/RUN. Per caricare, ci vogliono pochi secondi.

Nota: Un programma con una sola parte può essere registrato e ricaricato nello stesso modo oppure si possono usare CSAVE e CLOAD.

Nota: Ricordatevi di caricare il DOS prima di scrivere il programma.

MODIFICA DI UN PROGRAMMA BASIC SU DISCO

Riassumiamo nei punti seguenti il procedimento che permette di modificare un programma BASIC già esistente, registrato su dischetto:

1. Spegnere la console dell'ATARI ed inserire la cartuccia BASIC.
2. Collegare l'unità disco ed accenderla senza inserire il dischetto.
3. Aspettare che la luce si spenga e che il disco si fermi.
4. Inserire il dischetto (con il DOS) e chiudere lo sportello.
5. Accendere la console. Il DOS dovrebbe essere caricato e sullo schermo dovrebbe apparire READY.

6. Per caricare il programma dal disco scrivere
   LOAD "D:filename.ext"
7. Modificare il programma (o scrivere un programma nuovo).
8. Per conservare su disco il programma scrivere
   SAVE "D:filename.ext"
9. Aspettare sempre che si spenga la luce prima di rimuovere il dischetto.
10. Per ottenere la lista del direttorio, non rimuovere il dischetto e scrivere
    DOS
    Dopo aver premuto **[RETURN]**, apparirà il Menu DOS. Scegliere la lettera di comando **"A"**, scriverla e premere **[RETURN]** due volte per ottenere sullo schermo la lista del direttorio; oppure scrivere **"A"**, premere **[RETURN]** e poi **P: [RETURN]** per avere la lista del direttorio sulla stampante.
11. Per tornare al BASIC, scrivere **B [RETURN]**, oppure premere **[RESET]**.

CAPITOLO 6

BIBLIOTECA DI FUNZIONI

In questo capitolo descriveremo le funzioni aritmetiche e trigonometriche incluse nel BASIC ATARI ed altre funzioni che svolgono computi particolari. Una funzione, applicata a uno o più argomenti (valori numerici o stringhe) esegue un calcolo, il cui risultato (generalmente un numero), che diremo valore della funzione per l'argomento o argomenti dati, può essere direttamente riportato sullo schermo o può venir utilizzato per successive computazioni. Seguendo le convenzioni matematiche abituali, il BASIC indica il valore di una funzione, ad esempio della funzione COS, applicata all'argomento X con COS(X); l'argomento segue cioè il nome della funzione ed è racchiuso tra parentesi.

Nella sezione sulle funzioni trigonometriche parleremo anche di due istruzioni, RAD (radianti) e DEG (gradi), che si utilizzano in associazione con le funzioni trigonometriche. Ogni funzione descritta in questo capitolo può occorrere sia in un comando che in un'istruzione. Sono ammesse anche le funzioni multiple, cioè funzioni applicate a degli argomenti che, a loro volta, contengono qualche funzione.

In ciò che segue descriveremo le seguenti funzioni ed istruzioni:

ABS	**ATN**	**ADR**
CLOG	**COS**	**FRE**
EXP	**SIN**	**PEEK**
INT	**DEG/RAD**	**POKE**
LOG		**USR**
RND		
SGN		
SQR		

FUNZIONI ARITMETICHE

ABS

Formato: ABS (aexp)
Esempio: 100 AB = ABS (–190)

Questa funzione computa il valore assoluto di un numero positivo o negativo. Il valore risultante sarà dunque sempre un numero positivo.

CLOG

Formato: CLOG (aexp)
Esempio: 100 C = CLOG (83)

Computa il logaritmo in base 10 del suo argomento. CLOG (0) non dà un errore ma dà un risultato non attendibile, e CLOG (1), a causa di approssimazioni interne, può non dare esattamente 0.

EXP

Formato: EXP (aexp)
Esempio: 100 PRINT EXP (3)

Questa funzione computa il valore di *e* (numero di Neper, approssimativamente 2.71828283) elevato alla potenza specificata dall'argomento stesso della funzione (l'espressione tra parentesi). Nell'esempio dato sopra, il numero risultante sarà 20.0855365. In alcuni casi, la precisione del risultato è limitata alla prime sei cifre decimali.

INT

Formato: INT (aexp)
Esempi: 100 I = INT (3.445)
(alla variabile I verrà assegnato il valore 3)
100 X = INT (–14.66778)
(alla variabile X verrà assegnato il valore –15)

Il valore di questa funzione per un dato argomento è il più grande numero intero minore o eguale al valore dell'argomento. Ciò vale sia quando il valore dell'argomento è un numero positivo, sia quando è un numero negativo. Perciò, nel nostro primo esempio alla variabile I verrà assegnato il valore 3 e, nel secondo esempio, alla variabile X verrà assegnato il valore –15 (il primo numero intero minore o uguale a –14.66778). La funzione INT non deve essere confusa con quella usata sulle macchine calcolatrici, che opera semplicemente eliminando tutte le cifre decimali del valore del suo argomento.

LOG

Formato: LOG (aexp)
Esempio: 100 L = LOG (67.89/2.57)

Dà il logaritmo naturale del numero o del valore dell'espressione che ha come argomento. Per i valori di LOG (0) e LOG (1), vedi CLOG.

RDN

Formato: RND (aexp)
Esempio: 10 A = RND (0)

Il risultato di questa funzione sarà un numero compreso tra 0 e 1 (più precisamente, un numero maggiore o eguale a 0 e minore di 1), generato dall'hardware in maniera apparentemente casuale. La variabile o l'espressione tra parentesi che compare come argomento di RND è muta, cioè non ha alcun effetto sui numeri che risulteranno; essa deve comunque essere sempre presente. La funzione RND viene generalmente usata in combinazione con altre istruzioni o funzioni del BASIC per generare un numero da utilizzare in un gioco, o in un programma in cui si voglia far scegliere l'elaboratore stesso tra varie possibilità, etc. Quella che segue è una routine molto semplice che genererà un numero a caso compreso tra 0 e 999.

```
10 X = RND(0)
20 RX = INT(1000*X)
30 PRINT RX
```

(0 è una variabile muta).

SGN

Formato: SGN (aexp)
Esempio: 100 X = SGN(–199)
(Il valore di X sarà –1).

La funzione SGN darà il valore –1 se l'aexp ha come valore un numero negativo, 0 se l'aexp ha valore 0 e 1 se l'aexp ha come valore un numero positivo.

SQR

Formato: SQR(aexp)
Esempio: 100 PRINT SQR(100)
(Sullo schermo apparirà il numero 10).

Computa la radice quadrata dell'aexp, che dunque deve avere un valore positivo.

FUNZIONI TRIGONOMETRICHE

ATN

Formato: ATN(aexp)
Esempio: 100 X = ATN(65)

Computa l'arco tangente del suo argomento.

COS

Formato: COS(aexp)
Esempio: 100 C = COS(X+Y+Z)

Nota: si presume che X, Y e Z siano state definite precedentemente.
Dà il coseno trigonometrico dell'aexp.

SIN

Formato: SIN(aexp)
Esempio: 100 X = SIN(Y)

Nota: Si presume che Y sia stata definita precedentemente.

Dà il seno trigonometrico dell'argomento.

DEG/RAD

Formato: DEG
RAD
Esempi: 100 DEG
100 RAD

Queste due istruzioni permettono al programmatore di specificare se il computo delle funzioni trigonometriche debba essere effettuato su valori espressi in gradi o in radianti. L'elaboratore, se non ha ricevuto l'istruzione DEG, utilizza i radianti. Per tornare ai radianti dopo aver dato l'istruzione DEG, che ha provocato il passaggio ai computi in gradi, si deve dare l'istruzione RAD.

Vedere l'Appendice E per ulteriori funzioni trigonometriche che si possono definire in base a quelle date qui.

FUNZIONI SPECIALI

ADR

Formato: ADR(svar)
Esempio: ADR(A$)

Dà l'indirizzo di memoria decimale della stringa specificata dall'espressione tra parentesi. La conoscenza dell'indirizzo permette al programmatore di passare l'informazione a routines USR, etc. (vedi USR e Appendice B).

FRE

Formato: FRE(aexp)
Esempi: PRINT FRE (0)
100 IF FRE (0)<1000 THEN PRINT "MEMORIA CRITICA"

Questa funzione dà il numero di bytes liberi rimasti in RAM. Viene utilizzata principalmente in modo diretto, seguita da una variabile muta (0), perché il programmatore possa sapere quanto spazio di memoria gli resta per completare un dato programma. Naturalmente FRE si può usare anche in modo differito, all'interno di un programma BASIC.

PEEK

Formato: PEEK(aexp)
Esempi: 1000 IF PEEK(4000) = 255 THEN PRINT "255"
100 PRINT "IL MARGINE DI SINISTRA È": PEEK(82)

Fornisce il contenuto dell'indirizzo di memoria specificato dall'aexp. L'aexp deve essere un numero intero compreso tra 0 e 65535 o un'espressione aritmetica il cui valore sia un numero intero compreso tra 0 e 65535, che rappresenterà, in notazione decimale (non esadecimale), l'indirizzo di memoria di cui si vuole conoscere il contenuto. Il valore della funzione sarà anch'esso un numero intero in notazione decimale, compreso tra 0 e 255. Questa funzione permette all'utente di esaminare sia le posizioni RAM che quelle ROM. Il primo esempio sopra riportato utilizza PEEK per determinare se la posizione 4000 (in notazione decimale) contiene il numero 255. Nel secondo esempio, la funzione PEEK è usata per esaminare la posizione del margine di sinistra (vedi Appendice I).

POKE

Formato: POKE aexp1, aexp2
Esempi: POKE 82, 10
100 POKE 82, 20

Sebbene questa non sia una funzione, viene inclusa in questa sezione perché è strettamente collegata alla funzione PEEK. Il comando POKE inserisce dati in una locazione di memoria o modifica i dati in essa già contenuti. Nel formato dato sopra, aexp1 è l'indirizzo in notazione decimale della locazione da trovare e aexp2 è il dato da inserire in tale locazione.

Notare che il valore di aexp2 deve essere un numero intero, espresso in notazione decimale, compreso tra 0 e 255. POKE non può essere utilizzato per cambiare il contenuto di una locazione ROM. Per ottenere una certa familiarità con questo comando è opportuno cercare la locazione di memoria con un comando PEEK ed appuntarne il contenuto. Poi, se POKE non dà il risultato che si voleva, si può rimettere il vecchio contenuto nella locazione con un nuovo comando POKE.

L'esempio dato sopra di un comando POKE in modo diretto sposta il margine di sinistra dello schermo dalla sua normale posizione di 2 ad una posizione di 10. In altre parole, il margine verrà spostato di 8 spazi a destra. Per riportare il margine alla sua posizione normale, premere **[RESET]**.

USR

Formato: USR(aexp1 [,aexp2] [aexp3...])
Esempio: 100 RESULT = USR(ADD1,A*2)

Questa funzione dà il risultato di una subroutine in linguaggio macchina. La prima espressione, aexp1, deve essere un numero intero o un'espressione aritmetica il cui valore sia un numero intero rappresentante l'indirizzo di memoria, in notazione decimale, della routine in linguaggio macchina da eseguire. Gli argomenti di USR, aexp2, aexp3, etc., sono

opzionali. Essi dovrebbero essere espressioni aritmetiche il cui valore vari, in notazione decimale, da 0 a 65535. Si può anche usare un valore non intero, ma esso sarà arrotondato al numero intero più vicino.

Questi valori saranno convertiti dal formato di numero in virgola mobile del sistema decimale codificato binario (BCD, Binary Coded Decimal) del BASIC in un numero binario di due bytes. Questo viene poi mandato nello stack dell'harware, composto da un gruppo di locazioni di memoria RAM sotto il diretto controllo del chip microprocessore 6502. La figura 6 illustra la struttura dello stack dell'hardware.

N (numero degli argomenti nello stack– può essere 0)
X_1 (byte più significativo dell'argomento X)
X_2 (byte meno significativo dell'argomento X)
Y_1 (byte più significativo dell'argomento Y)
Y_2 (byte meno significativo dell'argomento Y)
Z_1 (byte più significativo dell'argomento Z)
Z_2 (byte meno significativo dell'argomentoZ)
•
•
•
R_1 (byte meno significativo dell'indirizzo di ritorno)
R_2 (byte più significativo dell'indirizzo di ritorno)

Figura 6-1. Definizione dello stack dell'hardware.

Nota: X è il primo argomento che segue l'indirizzo della routine, Y è il secondo, Z il terzo, etc. Ci sono N coppie di bytes.

Nel capitolo 11 descriveremo la funzione svolta da USR nella programmazione in linguaggio macchina. L'appendice D definisce i bytes in RAM disponibili per la programmazione in linguaggio macchina.

CAPITOLO 7

STRINGHE

In questo capitolo descriveremo le stringhe e le funzioni che permettono la manipolazione di stringhe. Ogni variabile stringa deve terminare con il simbolo $ e deve essere dimensionata prima di venir utilizzata in qualsiasi modo (vedere l'istruzione DIM nel capitolo 8). Una stringa è una successione di caratteri "legati" insieme, cioè considerati come un oggetto unico. I caratteri che compongono una stringa possono essere lettere, numeri, o altri simboli (compresi i simboli speciali della tastiera ATARI). Una sottostringa è una parte di una stringa più lunga e, nel BASIC Atari, è possibile accedere alle sottostringhe di una stringa data, se quest'ultima è stata appropriatamente dimensionata (vedere la parte finale di questo capitolo). I caratteri di una stringa sono numerati, andando da sinistra a destra; ad ognuno di essi viene cioè assegnato un indice che dipende dalla posizione che il carattere stesso occupa nella stringa: l'indice del primo carattere di una stringa è 1, quello del secondo è 2, e così via. Gli indici vanno quindi dal numero 1 al numero corrispondente alla lunghezza della stringa considerata; tale lunghezza sarà minore o eguale alla lunghezza con la quale la stringa è stata dimensionata.

Le funzioni da o su stringhe che descriveremo in questo capitolo sono:

ASC
CHR$
LEN
STR$
VAL

ASC

Formato: ASC(sexp)
Esempio: 100 A = ASC(A$)

Questa funzione, applicata ad una espressione stringa, dà il numero di codice ATASCII del primo carattere della stringa stessa. Si può usare sia in modo diretto che in modo differito. Nella fig. 7-1 un breve programma illustra l'uso della funzione ASC.

```
10 DIM A$(3)
20 A$="E"
30 A=ASC(A$)
40 PRINT A
```

Figura 7-1. Programma con la funzione ASC.

Nell'esecuzione di questo programma, apparirà sul video il numero 69, che è il codice ATASCII della lettera "E". Notate che le stringhe che occorrono in un'istruzione o un comando vanno sempre racchiuse tra virgolette (come nella riga 20 del programma precedente).

CHR$

Formato: CHR$(aexp)
Esempi: 100 PRINT CHR$(65)
100 A$ = CHR$(65)

Questa funzione, che va da numeri a stringhe, assegna ad ogni numero di codice ATASCII (aexp) la stringa composta da un unico carattere, quello rappresentato dal valore dell'aexp. Negli esempi dati sopra, il valore di CHR$(65) è la stringa composta soltanto dalla lettera A. Il programma che segue utilizza le funzioni ASC e CHR$ per stampare le lettere dell'alfabeto in caratteri maiuscoli e minuscoli.

```
10 FOR I=0 TO 25
20 PRINT CHR$(ASC("A")+I),CHR$(ASC("a")+I)
30 NEXT I
```

Figura 7-2. Esempio di programma con ASC e CHR$.

Nota: In una comparazione logica STR$ e CHR$ possono occorrere soltanto una volta ciascuno.

LEN

Formato: LEN(sexp)
Esempio: 100 PRINT LEN (A$)

Questa funzione ha come valore la lunghezza in bytes (cioè il numero di caratteri) della stringa designata dal suo argomento; potrete richiedere che tale valore venga scritto subito oppure utilizzarlo in un programma. La lunghezza di una stringa è semplicemente l'indice dell'ultimo carattere della stringa. Una stringa vuota (o meglio, la stringa vuota, che non contiene alcun carattere) o una variabile stringa alla quale non sia ancora stata assegnata alcuna stringa ha lunghezza 0. È anche possibile estendere una stringa già definita, cioè aggiungere al suo interno dei nuovi elementi, riferendosi opportunamente alle posizioni dei caratteri nella stringa per mezzo degli indici.

La seguente routine è un esempio di come si possa utilizzare la funzione LEN:

```
10 DIM A$(10)
20 A$ = "ATARI"
30 PRINT LEN(A$)
```

Figura 7-3. Esempio di programma con la funzione LEN.

Come risultato dell'esecuzione di questo programma, l'elaboratore stamperà il numero 5.

STR$

Formato: STR$(aexp)
Esempio: A$ = STR$(65)

Questa funzione va da numeri a stringhe: applicata ad una aexp dà, come risultato, la stringa il cui unico elemento è il valore dell'aexp. Nell'esempio dato sopra, STR$(65) sarà la stringa "65", sarà cioè lo stesso numero 65 che l'elaboratore riconoscerà però come stringa.

Nota: In una comparazione logica può occorrere soltanto un STR$ e soltanto un CHR$. Così, ad esempio, l'elaboratore non risponderà correttamente al comando A = STR$(1) >STR$(2).

VAL

Formato: VAL(sexp)
Esempio: 100 A = VAL(A$)

Questa funzione è, in un certo senso, l'inversa della funzione STR$: applicata ad una sexp della forma "n", dove n è un numero (eventualmente di più cifre), dà il numero n come valore. Se è applicata ad una stringa che inizia con un numero seguito da altri caratteri, VAL darà come valore soltanto il primo numero; così ad esempio, VAL("356+45") ha come valore 356 (e non 401). Potrete usare la funzione VAL per eseguire operazioni aritmetiche su stringhe, come illustra il seguente esempio:

```
10 DIM B$(5)
20 B$ = "10000"
30 B = SQR(VAL(B$))
40 PRINT "LA RADICE QUADRATA DI "; B$;" È "; B
```

Figura 7-4. Programma con la funzione VAL.

Eseguito il programma, sullo schermo apparirà la scritta: LA RADICE QUADRATA DI 10000 È 100.

Naturalmente non è possibile applicare la funzione VAL ad una stringa che non inizi con un numero o che l'elaboratore non possa interpretare come numero. Si può comunque applicare a stringhe che contengono numeri in virgola mobile; ad esempio, VAL("1E9") darà il numero 1.000.000.000.

MANIPOLAZIONE DI STRINGHE

Le stringhe possono venir manipolate in diversi modi: possiamo suddividerle, concatenarle, metterle in un determinato ordine. Nei paragrafi che seguono descriveremo questi diversi tipi di operazioni su stringhe.

CONCATENAZIONE DI STRINGHE

Concatenare due o più stringhe significa metterle assieme, una dopo l'altra, per formare una stringa più lunga, che conterrà come sottostringhe tutte le stringhe di partenza. Queste ultime devono essere state prima dimensionate (vedi DIM). La stringa risultante da una concatenazione può essere assegnata ad una nuova variabile stringa, o può venir stampata

direttamente, oppure può essere utilizzata in una parte successiva del programma. Osservando il programma della figura7-5, si comprenderà quale forma debba avere un'istruzione di concatenazione di stringhe. Le stringhe A$, B$ e C$ vengono qui concatenate e il risultato della concatenazione è assegnato alla variabile A$. (Anticipiamo qui, ma lo spiegheremo meglio nel prossimo paragrafo, che A$(aexp) è la sottostringa di A$ che inizia con il carattere con indice aexp e termina con l'ultimo carattere della stringa A$).

```
10 DIM A$(100), B$(100), C$(100)
20 A$ = "NEL CAPITOLO 9 DI "
30 B$ = " 'BASIC ATARI- -GUIDA AUTODIDATTICA' "
40 C$ = "SI PARLA DI STRINGHE E SOTTOSTRINGHE"
50 A$(LEN(A$)+1)=B$
60 A$(LEN(A$)+1)=C$
70 PRINT A$
```

Figura 7-5. Esempio di concatenazione di stringhe.

SUDDIVISIONE DI STRINGHE

Se vogliamo suddividere una stringa, cioè prelevarne una determinata sottostringa, utilizziamo il formato delle variabili stringa indicizzate. Il formato di una variabile stringa indicizzata è il seguente:

svarname(aexp1[,aexp2])

svarname è il nome della stringa che si vuole suddividere, cioè una variabile stringa; aexp1 indica l'inizio della sottostringa (precisamente, la posizione o l'indice, nella stringa di partenza, di quello che sarà il primo carattere della sottostringa) e aexp2 — che può anche mancare — indica la fine della sottostringa (la posizione dell'ultimo carattere della sottostringa); cioè svarname(aexp1,aexp2) è la sottostringa della stringa indicata da svarname, che inizia con il carattere il cui indice è aexp1 e termina con il carattere con indice aexp2. Se l'aexp2 non è specificata, allora la fine della sottostringa corrisponde alla fine della stringa stessa. Naturalmente aexp1 non deve essere maggiore della lunghezza della stringa (a meno che il formato non venga utilizzato, come nel paragrafo precedente, per la concatenazione di stringhe). I due programmi della figura 7-6 illustrano l'operazione di suddivisione di una stringa con o senza indicazione della fine della sottostringa.

```
10 DIM S$(5)
20 S$="ABCD#"
30 PRINT S$(2)
40 END
```

```
10 DIM S$(20)
20 S$="BASIC ATARI 800"
30 PRINT S$(7,11)
40 END
```

Figura 7-6. Esempi di suddivisione di stringhe.

Come risultato dell'esecuzione del primo programma, in cui non è indicata la fine della sottostringa, sul video apparirà BCD#. Nel secondo programma invece è indicata la fine della sottostringa che risulterà dalla suddivisione, cioè della sottostringa "ATARI".

COMPARAZIONE E ORDINAMENTO DI STRINGHE

Nel comparare due stringhe gli operatori logici funzionano esattamente nello stesso modo in cui operano con i numeri. Il secondo programma riportato nell'appendice H è un esempio molto semplice di "ordinamento a bolla" (bubble sort).

Nell'usare gli operatori logici, ricordate che ad ogni lettera, numero o simbolo è assegnato un numero di codice ATASCII. A questi numeri di codice si applicano alcune regole generali:

1. I codici ATASCII per i numeri corrispondono ai valori reali dei numeri e sono sempre minori dei codici delle lettere (vedere l'appendice C).
2. Le lettere maiuscole hanno valori numerici minori delle lettere minuscole. Per ottenere il codice ATASCII di una lettera minuscola, conoscendo il numero di codice della lettera maiuscola, si aggiunge 32 al codice maiuscolo.

Nota: Il sistema di governo della memoria del BASIC Atari può cambiare la locazione di una stringa nella memoria per far posto a nuove istruzioni. Perciò, se un programma viene modificato o se si usa il modo diretto, l'indirizzo delle stringhe può cambiare.

VETTORI E MATRICI

Un vettore è una lista unidimensionale di variabili, ognugna delle quali è designata da un nome di variabile, che è lo stesso per tutte le variabili del vettore, ed un indice (un valore numerico che segue, tra parentesi, il nome della variabile) che la distingue dalle altre variabili del vettore; ad esempio A(0), A(1), A(2) è un vettore. Gli indici variano da 0 al valore dimensionato (vedi l'istruzione DIM). La figura 8-1 illustra un vettore con 7 elementi.

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)

Figura 8-1. Esempio di vettore.

Una matrice o vettore bidimensionale è, in questo contesto, una tavola bidimensionale che contiene righe e colonne. Le righe vanno in senso orizzontale, le colonne in senso verticale. Ad ogni punto d'incrocio tra una riga e una colonna corrisponde un elemento della matrice. Il BASIC conserva gli elementi di una matrice in ordine crescente di riga; ciò significa che tutti gli elementi della prima riga sono immagazzinati per primi, seguiti da quelli della seconda, etc. La fig. 8-2 illustra una matrice 7*4 (di 7 colonne per 4 righe).

Colonne

Righe				
	M(0,0)	M(0,1)	M(0,2)	M(0,3)
	M(1,0)	M(1,1)	M(1,2)	M(1,3)
	M(2,0)	M(2,1)	M(2,2)	M(2,3)
	M(3,0)	M(3,1)	M(3,2)	M(3,3)
	M(4,0)	M(4,1)	M(4,2)	M(4,3)
	M(5,0)	M(5,1)	M(5,2)	M(5,3)
	M(6,0)	M(6,1)	M(6,2)	M(6,3)

Figura 8-2. Esempio di matrice.

In questo capitolo descriveremo i due comandi associati a vettori, matrici e stringhe ed indicheremo inoltre come assegnare valori a vettori e matrici. I comandi di cui parleremo sono:

DIM
CLR

DIM (DI.)

Formato: DIM { svar(aexp) | mvar(aexp[,aexp]) } [{ ,svar(aexp) | ,mvar(aexp[,aexp...] }]

Esempi:

```
DIM A(100)
DIM M(6,3)
DIM B$(20)            usato con le STRINGHE
```

Prima di assegnare un valore ad una variabile stringa o di costruire un vettore o una matrice, essi devono essere dimensionati, dobbiamo cioè informare l'elaboratore di quanto spazio di memoria dovrà riservare agli elementi che definiremo. L'istruzione DIM serve appunto per riservare ad una stringa, un vettore o una matrice un certo numero di locazioni di memoria (ogni carattere di una stringa occupa un byte di memoria, ed ogni elemento di un vettore occupa sei bytes).

Quando si dimensiona una variabile stringa con un'istruzione DIM A$ (aexp), il valore dell'aexp deve essere maggiore o eguale alla lunghezza massima che potrà avere la stringa assegnata ad A$ (vedi LEN nel capitolo 7). Se dimensioniamo un vettore con DIM X (aexp), significa che il vettore designato da X conterrà al massimo aexp + 1 elementi. Infine se dimensioniamo una matrice con DIM M (aexp1, aexp2), la matrice non avrà più di aexp1 + 1 righe e aexp2 + 1 colonne. Così il primo esempio dato sopra riserva al vettore indicato da A uno spazio di memoria sufficiente per 101 elementi. Il secondo esempio riserva 28 locazioni di memoria per il vettore bidimensionale (matrice) indicato da M. Il terzo esempio riserva 20 bytes per la stringa indicata da B$. Inoltre il vettore della figura 8-1 può essere dimensionato con DIM A(6) anche se in realtà contiene 7 elementi per via della presenza dell'elemento A(0). Analogamente, la matrice della figura 8-2 può essere dimensionata con DIM M(6,3) e tuttavia il BASIC riserverà per essa uno spazio di memoria sufficiente a contenere 28 elementi.

Tutte le variabili stringa, i vettori e le matrici devono essere dimensionati prima di essere definiti. E' abituale, e conveniente, mettere tutte le istruzioni DIM all'inizio del programma.

Nota: L'elaboratore personale ATARI non inizializza automaticamente a 0 variabili vettore o variabili matrice, all'inizio dell'esecuzione del programma. Per inizializzare a 0 gli elementi di un vettore o di una matrice, potete ad esempio includere nel programma le seguenti istruzioni:

```
250 DIM A(100)
300 FOR E=0 TO 100
310 A(E)=0
320 NEXT E
```

I vettori e le matrici vengono "riempiti" di dati usando istruzioni FOR/NEXT, istruzioni READ/DATA o comandi INPUT. La figura 8-3 illustra la costruzione di un vettore per mezzo di un ciclo FOR/NEXT, e la figura 8-4, la costruzione di un vettore che utilizza le istruzioni READ/DATA.

```
10 DIM A(90)
20 X=10
30 FOR E=0 TO 90
40 X=X+1
50 A(E)=X
60 NEXT E
70 FOR E=0 TO 90
80 PRINT E,A(E)
90 NEXT E
```

Figura 8-3. Uso di FOR/NEXT per costruire un vettore.

```
10 DIM A(3)
20 FOR E=0 TO 3
30 READ X
40 A(E)=X
50 PRINT A(E),
60 NEXT E
70 END
100 DATA 33,45,12,58
```

Figura 8-4. Uso di READ/DATA per costruire un vettore.

Nella figura 8-5 viene costruita una matrice 7*3.

```
10 DIM M(6,3)
20 FOR RIGA=0 TO 6
30 FOR COL=1 TO 3
40 M(RIGA,COL)=INT(RND(0)*1000)
50 NEXT COL:NEXT RIGA
60 FOR RIGA=0 TO 6
70 FOR COL=1 TO 3
80 PRINT M(RIGA,COL)
90 NEXT COL:PRINT :NEXT RIGA
```

Figura 8-5. Costruzione di una matrice.

Notiamo che le parole RIGA e COL non sono comandi BASIC, né istruzioni, né funzioni, né parole-chiave. Sono semplicemente dei nomi di variabile, usati qui per indicare qual è il ciclo interno e quale quello esterno. Il programma sarebbe stato essenzialmente lo stesso se avessimo utilizzato come nomi di variabili X e Y.

CLR

Formato: CLR
Esempio: 200 CLR

Questo comando cancella dalla memoria tutte le variabili stringa, i vettori e le matrici precedentemente dimensionati, cosicché si possono utilizzare per altri scopi gli stessi nomi di variabili e lo spazio di memoria precedentemente occupato. Cancella anche i valori assegnati a variabili non dimensionate (variabili aritmetiche non sottoscritte). Se dopo un comando CLR vogliamo reintrodurre una matrice, un vettore o una stringa, dovremo dimensionarli con un nuovo comando DIM.

MODALITÀ E COMANDI GRAFICI

In questo capitolo illustreremo le diverse modalità grafiche dell'elaboratore personale ATARI ed i relativi comandi del BASIC Atari. Per mezzo di questi comandi è possibile creare schemi grafici e disegni, ad esempio per movimentare un gioco con illustrazioni e colori.

Nel seguito descriveremo i seguenti comandi:

GRAPHICS	**LOCATE**	**PUT/GET**
COLOR	**PLOT**	**SETCOLOR**
DRAWTO	**POSITION**	**XIO**

I comandi PUT/GET e XIO di cui parleremo qui rappresentano applicazioni particolari degli stessi comandi descritti nel capitolo 5.

GRAPHICS (GR.)

Formato: GRAPHICS aexp

Esempio: GRAPHICS 2

Questo comando serve per scegliere una delle nove modalità grafiche di cui il BASIC Atari dispone. La tabella 9-1 illustra le caratteristiche di ognuna di esse. Il comando GRAPHICS apre automaticamente lo schermo, S: (la "finestra della grafica"), come dispositivo numero 6. Perciò quando si vuole scrivere sulla "finestra del testo", non è necessario specificare il codice del dispositivo. L'aexp, arrotondata all'intero più vicino, deve dare un numero positivo. Nella modalità grafica 0 si ha una visualizzazione "a schermo intero" (full-screen), mentre le modalità 1-8 danno una visualizzazione a "schermo suddiviso" (split-screen) in due parti: la finestra della grafica, in alto, e la finestra del testo in basso. Perché una modalità normalmente a schermo suddiviso appaia invece a schermo intero (con lo schermo occupato interamente dalla finestra della grafica, quando vengono eseguiti comandi grafici, e con lo schermo interamente occupato dalla finestra del testo, quando vengono eseguiti comandi di "scrittura"), nel comando GRAPHICS aggiungete i caratteri + 16 al numero di modalità (aexp). Per evitare che il comando GRAPHICS cancelli la parte di memoria riservata alla finestra della grafica, aggiungete invece i caratteri + 32.

Per tornare alla modalità grafica 0 in modo diretto, premere **[RESET]**, oppure scrivere GR.0 e premere **[RETURN]**.

Tabella 9-1. Modalità e formati dello schermo

Modo grafico	Tipo	FORMATO DELLO SCHERMO Colonne	Righe- (schermo diviso)	Righe- (schermo pieno)	Numero colori	Memoria RAM richiesta (bytes) divisa	piena
0	TEXT	40	—	24	1-1/2		992
1	TEXT	20	20	24	5	674	672
2	TEXT	20	10	12	5	424	420
3	GRAPHICS	40	20	24	4	434	432
4	GRAPHICS	80	40	48	2	694	696
5	GRAPHICS	80	40	48	4	1174	1176
6	GRAPHICS	160	80	96	2	2174	2184
7	GRAPHICS	160	80	96	4	4190	4200
8	GRAPHICS	320	160	192	1-1/2	8112	8138
9	GRAPHICS	80	—	192	1		8138
10	GRAPHICS	80	—	192	9		8138
11	GRAPHICS	80	—	192	16		8138
12	GRAPHICS	40	20	24	5	1154	1152
13	GRAPHICS	40	10	12	5	664	660
14	GRAPHICS	160	160	192	2	4270	4296
15	GRAPHICS	160	160	192	4	8112	8138

Nei paragrafi che seguono descriveremo le nove modalità grafiche.

MODALITÀ GRAFICA 0

Questa modalità, a colore 1 e luminosità 2, è la modalità per difetto (quella che vale al momento dell'accensione) dell'elaboratore personale ATARI. Ha una matrice di schermo di 24 righe per 40 colonne. La posizione per difetto dei margini è alla seconda e trentanovesima colonna, cosicché si possono avere 38 caratteri in ogni riga. Potete modificare i margini per mezzo di un comando POKE che cambi il contenuto delle locazioni di memoria LMARGIN o RMARGIN (indirizzi numero 82 e 83, vedi l'appendice I). Il colore dei caratteri è determinato dal colore dello sfondo. Può variare solo la loro luminosità. La visualizzazione a schermo intero ha sfondo blu bordato in nero (a meno che non si specifichi che il bordo debba essere di un altro colore). Se volete scrivere un carattere in una posizione specifica sullo schermo, potete ricorrere ad uno dei metodi seguenti:

Metodo 1.

lineno POSITION aexp1, aexp2	*Porta il cursore nella posizione specificata da aexp1 e aexp2.*
lineno PRINT sexp	

Metodo 2.

lineno GR.0	*Specifica la modalità grafica*
lineno POKE 752,1	*Sopprime il cursore.*
lineno COLOR ASC(sexp)	*Specifica il carattere da stampare.*
lineno PLOT aexp1,aexp2	*Specifica dove stampare il carattere.*
lineno GOTO lineno	*Dà inizio ad un ciclo per evitare che venga stampato READY e soppressa la finestra della grafica (l'istruzione rimanda al suo stesso numero di riga).*

Premere **[BREAK]** *per interrompere il ciclo.*

GRAPHICS 0 si può anche usare, in modo diretto o differito, come comando per cancellare tutto ciò che appare sullo schermo. Mette fine a qualsiasi modalità grafica precedentemente scelta e riporta lo schermo alla modalità per difetto (GRAPHICS 0).

MODALITÀ GRAFICHE 1 E 2

Queste due modalità a 5 colori sono, come si può vedere nella tabella 9-1, modalità di testo, ma a schermo suddiviso (vedi figura 9-1). Nella modalità grafica 1 i caratteri hanno larghezza doppia, ma la stessa altezza, rispetto a quelli stampati nella modalità grafica 0. Nella modalità grafica 2 i caratteri hanno larghezza ed altezza doppie rispetto a quelli della modalità 0. Nelle modalità a schermo suddiviso, il comando PRINT serve per far apparire i caratteri sia nella finestra del testo che nella finestra della grafica. Per scrivere sulla finestra della grafica, specificare il dispositivo numero 6 (# 6) dopo il comando PRINT.

Esempio:

```
100 GR.1
110 PRINT #6; "ATARI"
```

I calori per difetto dipendono dal tipo dei caratteri in ingresso. La tabella 9-2 mostra il colore per difetto ed il registro di colore usato per ogni tipo di carattere.

Tabella 9-2. Colori per difetto dei vari tipi di caratteri in ingresso

Tipo di carattere	Registro di colore	Colore per difetto
Lettere maiuscole	0	Arancione
Lettere minuscole	1	Verde chiaro
Lettere maiuscole video inverse	2	Blu scuro
Lettere minuscole video inverse	3	Rosso
Numeri	0	Arancione
Numeri video inversi	2	Blu scuro

Nota: Vedi **SETCOLOR** per cambiare i colori dei caratteri.

Finché non viene specificato altrimenti, tutti i caratteri appaiono in maiuscolo e non video inversi, cioè chiari su fondo scuro. Per scrivere le lettere minuscole o i caratteri grafici speciali della tastiera dell'ATARI, date il comando **POKE 756,226.** Per tornare a scrivere in maiuscolo, usate **POKE 756,224.**

Nelle modalità grafiche 1 e 2 non è possibile ottenere l'inversione figura-sfondo sul video, ma i caratteri possono essere stampati in quatto colori differenti (vedi la fine del capitolo).

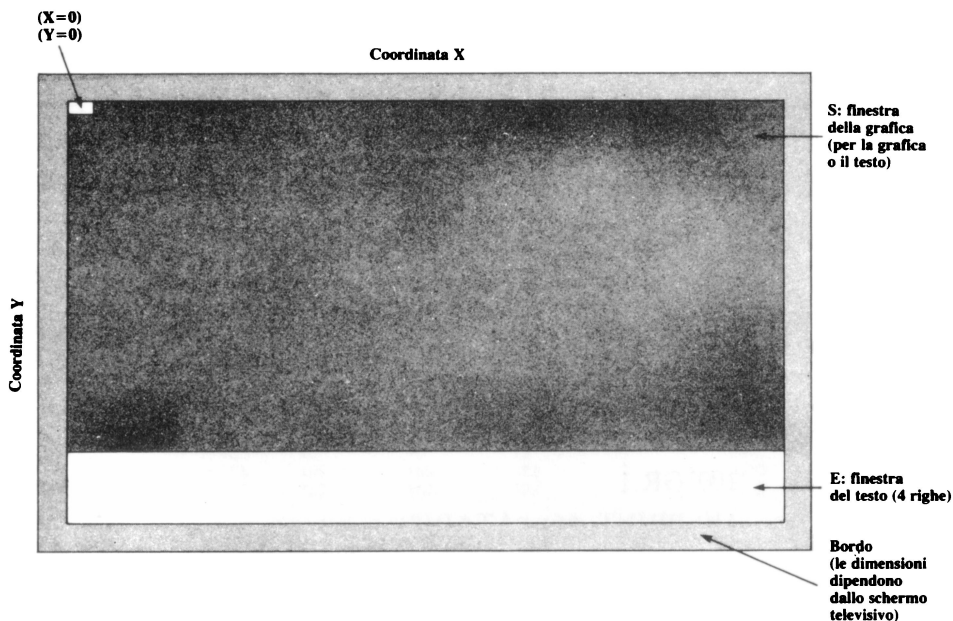



Figura 9-1. Visualizzazione a schermo suddiviso nelle modalità grafiche 1 e 2.

Le coordinate X e Y partono dal valore 0 (angolo superiore sinistro dello schermo). I valori massimi sono uguali al numero di righe o di colonne meno 1 (vedi la tabella 9-1).

Le modalità 1 e 2 si possono avere anche a schermo intero: basta aggiungere i caratteri +16 al numero di modalità, nel comando GRAPHICS.

Esempio: GRAPHICS 1 + 16

MODALITÀ GRAFICHE 3,5,7

Anche queste tre modalità grafiche, a quattro colori, danno una visualizzazione a schermo suddiviso, se non viene specificato altrimenti; si può comunque ottenere la visualizzazione a schermo intero aggiungendo +16 al numero di modalità che segue il comando GRAPHICS. Le modalità 3, 5 e 7 sono simili, tranne per il fatto che la 5 e la 7 dispongono di più punti o "pixels", sullo schermo, come possibili posizioni del cursore, per tracciare diagrammi o disegni; i punti sono più piccoli e danno quindi una più alta risoluzione grafica.

MODALITÀ GRAFICHE 4,6

Queste due modalità grafiche danno la visualizzazione a schermo suddiviso e dispongono soltanto di due colori, al contrario delle altre modalità con le quali si possono avere 4 o 5 colori. Il vantaggio delle modalità a due colori consiste nel fatto che esse richiedono un minore spazio di memoria RAM (vedi la tabella 9-1). Si usano perciò quando è sufficiente

disporre di due colori soltanto e quando la memoria RAM deve contenere parecchi dati. La grafica tracciata in queste due modalità ha una risoluzione più alta, data dai punti più piccoli, rispetto alla modalità grafica 3.

MODALITÀ GRAFICA 8

È la modalità che permette la più alta risoluzione grafica. Questa caratteristica però ha come contropartita l'occupazione di un maggiore spazio di memoria RAM; quindi, per bilanciare l'occupazione di spazio di memoria rispetto alle altre modalità, la modalità grafica 8 dispone soltanto di un colore e di due gradi diversi di luminosità.

COLOR (C.)

Formato: COLOR aexp
Esempi: 110 COLOR ASC ("A")
110 COLOR 3

Nel comando COLOR, il valore dell'espressione determina il dato da conservare nella memoria della grafica (Display memory) per tutti i successivi comandi PLOT e DRAWTO, finché non viene eseguito il successivo comando COLOR. Il valore dell'aexp deve essere positivo, ed è generalmente un numero intero compreso tra 0 e 255. I numeri non interi sono arrotondati all'intero più vicino. L'hardware della grafica interpreta questo dato in modi diversi nelle varie modalità grafiche. Nelle modalità per testo, 0, 1 e 2, il numero può essere compreso tra 0 e 255 (8-bits) e determina il carattere da stampare ed il suo colore. (I due bits più significativi determinano il colore. È per questo motivo che in queste modalità sono disponibili solo 64 caratteri differenti invece dell'insieme completo di 256 caratteri).

Le tabelle 9-6 e 9-7 alla fine del capitolo illustrano l'insieme dei caratteri interni e l'assegnazione carattere/colore. La tabella 9-2 è una tabella semplificata che permette di generare più facilmente alcuni colori. Per esempio, **COLOR ASC ("A"):PLOT 5,5**, nelle modalità grafiche 1 e 2, farà apparire un carattere A arancione nella posizione 5,5.

Le modalità grafiche da 3 a 8 non sono modalità per testo, per cui il dato conservato nella RAM della grafica determina semplicemente il colore di ogni pixel. Nelle modalità a due colori o a due luminosità vi saranno solo due valori significativi dell'aexp, 0 e 1 (1-bit), mentre nelle modalità a quattro colori ve ne saranno quatto: 0, 1, 2 e 3. (L'espressione nel comando COLOR può avere un valore maggiore di 3, ma verranno comunque utilizzati solo uno o due bits). Il colore che appare sullo schermo dipende dal valore conservato nel registro di colore che corrisponde ai dati 0, 1, 2 o 3 nella particolare modalità grafica che si sta utilizzando. Per conoscere la corrispondenza tra valori dell'aexp, registri e colori, si può consultare la tabella 9-5 che riporta i colori per difetto ed i corrispondenti numeri di registro. I colori si possono cambiare utilizzando un comando SETCOLOR.

Si deve notare che quando viene caricato il BASIC, il dato di colore è 0, e quando viene eseguito un comando GRAPHICS (senza i caratteri aggiuntivi +32), tutti i pixels contengono il dato 0. Perciò, nelle modalità grafiche 3-7, sembra che i comandi PLOT e DRAWTO non abbiano alcun effetto, se non è stato eseguito prima un comando COLOR. È opportuno dunque dare innanzitutto il comando COLOR 1.

DRAWTO (DR.)

Formato: DRAWTO aexp1, aexp2

Esempio: 100 DRAWTO 10, 8

Eseguendo questo comando il BASIC traccia una linea retta che va dall'ultimo punto apparso in seguito ad un comando PLOT (vedi **PLOT**) alla posizione specificata da aexp1 e aexp2. La prima espressione rappresenta la coordinata orizzontale X (la colonna) e la seconda rappresenta la coordinata verticale Y (la riga) del punto che coinciderà con l'estremità della linea (vedere la figura 9-1). La linea sarà dello stesso colore del punto apparso con l'ultimo comando PLOT.

LOCATE (LOC.)

Formato: LOCATE aexp1, aexp2, avar

Esempio: 150 LOCATE 12, 15, X

Questo comando porta il cursore grafico, che è invisibile, alla posizione, sulla finestra della grafica, specificata dall'aexp1 e dall'aexp2, trova il dato conservato in quel pixel e lo assegna alla variabile aritmetica specificata. Questa assumerà dunque un valore compreso tra 0 e 255 nelle modalità grafiche 0-2; avrà valore 0 o 1 nelle modalità grafiche a 2 colori; e 0, 1, 2 o 3 nelle modalità grafiche a quattro colori. Le due espressioni aritmetiche specificano le coordinate X e Y del punto su cui si sposterà il cursore grafico. LOCATE equivale a:

POSITION aexp1,aexp2:GET #6, avar

L'esecuzione di un comando PRINT dopo quella di un LOCATE o di un GET dallo schermo (cioè un GET #6) può provocare la modifica del dato presente nel pixel preso in esame. Si può risolvere il problema riposizionando il cursore e collocando di nuovo il dato letto nel pixel in questione; dopo aver fatto ciò si può tranquillamente dare il comando PRINT. Il programma seguente illustra l'uso del comando LOCATE.

```
10 GRAPHICS 3+16
20 COLOR 1
30 SETCOLOR 2,10,8
40 PLOT 10,15
50 DRAWTO 15,15
60 LOCATE 12,15,X
70 PRINT X
```

Figura 9-2. Esempio: programma con LOCATE.

Eseguendo il programma, il BASIC scriverà sulla finestra del testo il dato (1) che, secondo l'istruzione COLOR, è conservato nel pixel 12, 15.

PLOT (PL.)

Formato: PLOT aexp1, aexp2

Esempio: 100 PLOT 5,5

Il comando PLOT si usa nelle modalità grafiche 3-8 per far apparire un punto colorato nella finestra della grafica. L'aexp1 specifica la coordinata orizzontale e l'aexp2 la coordinata verticale della posizione in cui dovrà comparire il punto. Il suo colore è determinato dal colore e dalla luminosità conservati nel registro di colore, secondo l'ultima istruzione COLOR eseguita. Per cambiare il contenuto del registro di colore, e quindi il colore del punto tracciato, usare SETCOLOR. Il numero di punti che si possono tracciare sullo schermo dipende dalla modalità grafica usata. I valori possibili per le coordinate di un punto sono compresi tra 0 e il numero totale di righe meno 1, per la coordinata verticale, o il numero totale di colonne meno 1, per la coordinata orizzontale (vedere la tabella 9-1).

POSITION (POS.)

Formato: POSITION aexp1, aexp2

Esempio: 100 POSITION 8, 12

L'istruzione POSITION si usa per portare il cursore invisibile della finestra della grafica ad una posizione specifica dello schermo (generalmente precede un comodo PRINT). Quest'istruzione può essere usata in tutte le modalità. È interessante notare che il cursore in realtà non si muove finché non viene emesso un comando I/O che interessi lo schermo.

PUT/GET (PU./GE.)

Formato: PUT #aexp, aexp
GET #aexp, avar

Esempi: 100 PUT #6, ASC ("A")
200 GET #1, X

Lavorando con la grafica, PUT si usa come comando di uscita diretto allo schermo. Quest'istruzione è sempre in connessione con POSITION. Dopo l'esecuzione di un comando PUT (o GET) il cursore si sposta, sullo schermo, alla posizione successiva. Dando un comando PUT diretto al dispositivo #6 (lo schermo), alla posizione del cursore apparirà l'input di un byte che è specificato dalla seconda aexp. Il byte può essere quello di un numero di codice ATASCII di un dato carattere (modalità 0-2), oppure un dato di colore (modalità 3-8).

GET si usa per assegnare alla variabile aritmetica specificata (avar) il byte del numero di codice del carattere che si trova alla posizione del cursore. La corrispondenza tra valore dell'aexp e i caratteri o i colori utilizzati per un'istruzione PUT o GET è la stessa di quella utilizzata per l'istruzione COLOR (si possono anche usare PRINT e INPUT).

Dobbiamo qui ricordare quanto già notato a proposito di un comando GET dallo schermo seguito da un comando PRINT: è possibile che questa successione di comandi modifichi i dati presenti nel pixel preso in esame (vedere **LOCATE**).

SETCOLOR (SE.)

Formato: SETCOLOR aexp1, aexp2, aexp3
Esempio: 100 SETCOLOR 0, 1, 4

Questa istruzione serve per scegliere il colore e la luminosità particolari che si vogliono conservare in un dato registro di colore. I parametri dell'istruzione SETCOLOR sono definiti come segue:

aexp1 = Registro di colore (0-4, a seconda della modalità grafica);
aexp2 = Numero del colore (0-15. Vedere la tabella 9-3);
aexp3 = Luminosità del colore (deve essere un numero pari compreso tra 0 e 14. Il video sarà tanto più luminoso quanto più grande è il numero. 14 corrisponde quasi ad un bianco puro).

Tabella 9-3. Colori e numeri corrispondenti per i comandi SETCOLOR

COLORI	NUMERI PER SETCOLOR (aexp2)
GRIGIO	0
ARANCIONE CHIARO (ORO)	1
ARANCIONE	2
ROSSO ARANCIATO	3
ROSA	4
BLU VIOLETTO	6
BLU	7
BLU	8
AZZURRO CHIARO	9
TURCHESE	10
VERDE MARE	11
VERDE	12
VERDE-GIALLO	13
VERDE ARANCIATO	14
ARANCIONE CHIARO	15

Nota: I colori possono variare con il tipo e la regolazione del monitor televisivo.

L'hardware ATARI della grafica contiene cinque registri di colore, numerati da 0 a 4. Il sistema operativo (OS) dispone di cinque locazioni RAM (da COLOR0 a COLOR4, vedere l'appendice I – Locazioni di Memoria), in cui conserva il valore dei colori che sta usando. Per cambiare i valori residenti in queste locazioni RAM si usa l'istruzione SETCOLOR. (L'OS trasferisce tali valori ai registri hardware della struttura televisiva collegata all'elaboratore). Nell'istruzione SETCOLOR il registro di colore viene specificato da un valore che va da 0 a 4. Nelle istruzioni COLOR si usano numeri differenti perché i dati specificati corrispondono solo indirettamente ai registri di colore. Poiché tutto ciò potrebbe, in un primo momento, risultare poco chiaro, si consiglia di studiare e sperimentare accuratamente le tabelle di questo capitolo.

Se si usa l'insieme dei cinque colori per difetto, non è necessario alcun comando SETCOLOR. Sebbene siano possibili 128 combinazioni differenti di colore e luminosità, sul video non possono apparire contemporaneamente più di cinque colori alla volta, dato che il sistema operativo dispone soltanto di cinque registri di colore. Lo scopo dei registri di colore e dell'istruzione SETCOLOR è quello di specificare le cinque combinazioni da utilizzare.

Tabella 9-4. Colori per difetto di SETCOLOR *

Setcolor (registro di colore)	Numero del colore per difetto	Luminosità	Colore effettivo
0	2	8	ARANCIONE
1	12	10	VERDE
2	9	4	BLU SCURO
3	4	6	ROSA O ROSSO
4	0	0	NERO

* Il valore per difetto, come abbiamo più volte osservato, è quello che si ha in mancanza di istruzioni SETCOLOR.

Nota: I colori possono variare a seconda del tipo, della condizione e della regolazione del monitor televisivo.

Il programma che segue illustra l'uso della modalità grafica 3 e dei comandi già esposti nel corso di questo capitolo.

```
10 GRAPHICS 3
20 SETCOLOR 0,2,8:COLOR 1
30 PLOT 17,1:DRAWTO 17,10:DRAWTO 9,18
40 PLOT 19,1:DRAWTO 19,18
50 PLOT 20,1:DRAWTO 20,18
60 PLOT 22,1:DRAWTO 22,10:DRAWTO 30,18
70 POKE 752,1
80 PRINT :PRINT "          ELABORATORI PERSONALI ATARI"
90 GOTO 90
```

Le istruzioni SETCOLOR e COLOR stabiliscono il colore dei punti da tracciare (vedere la tabella 9-5). Il comando SETCOLOR carica il registra di colore 0 con il colore 2 (arancione) e la luminosità 8 ("normale "). Le quattro righe successive determinano i punti da tracciare. La riga 70 sopprime il cursore e la riga 80 stampa l'espressione stringa ELABORATORI PERSONALI ATARI nella finestra del testo, spostata di sei spazi a destra.

Notare che non è stato assegnato alcun colore allo sfondo, dal momento che il colore per difetto è quello desiderato (nero).

Una volta eseguito il programma, nella finestra della grafica apparirà il simbolo "Atari Logo" e nella finestra del testo l'espressione stringa, come illustra la figura 9-3.

XIO(X.) APPLICAZIONE SPECIALE DI RIEMPIMENTO

Formato: XIO 18, # aexp, aexp1, aexp2, filespec

Esempio: 100 XIO 18, #6,0,0,"S:"

Questa applicazione speciale del comando XIO serve a riempire un'area dello schermo compresa tra punti e linee, precedentemente tracciati, con un colore di valore diverso da zero. Come aexp1 e aexp2 si usano qui variabili mute (0).

I punti seguenti illustrano le operazioni da eseguire.

1. Dare un'istruzione PLOT diretta all'angolo inferiore destro (punto 1).
2. Dare un'istruzione DRAWTO diretta all'angolo superiore destro (punto 2). Così si è definito il margine destro dell'area da riempire.
3. Dare un'istruzione DRAWTO diretta all'angolo superiore sinistro (punto 3).
4. Con un POSITION, posizionare il cursore nell'angolo inferiore sinistro (punto 4).
5. Con un comando POKE, immettere il dato relativo al colore di riempimento (1, 2 o 3) nell'indirizzo 765.
6. Con questo metodo ogni riga orizzontale dell'area delimitata viene riempita di colore, procedendo all'alto verso il basso. L'operazione di riempimento comincia da sinistra e procede verso destra lungo la riga finché non raggiunge un atomo grafico che contiene un dato diverso da zero (quando è necessario, il riempimento "va a capo" e continua alla riga successiva). Ciò significa che non si può ricorrere a questa operazione per cambiare il colore di un'area che sia già stata riempita con un valore diverso da zero, perché l'operazione di riempimento si interromperebbe immediatamente. Se si cerca di riempire con dato zero (0) una riga che non contiene alcun atomo grafico con un dato diverso da zero, il comando di riempimento provocherà un ciclo senza fine. Se ciò accade, si possono premere i tasti **[BREAK]** o **[RESET]** per fermare l'operazione di riempimento.

Il programma seguente disegna una figura e la riempie con il dato (colore) 3. Notare che il comando XIO traccia automaticamente il margine inferiore e quello di sinistra della figura.

```
10 GRAPHICS 5+16
20 COLOR 3
30 PLOT 70,45
40 DRAWTO 50,10
50 DRAWTO 30,10
60 POSITION 10,45
70 POKE 765,3
80 XIO 18,#6,0,0,"S:"
90 GOTO 90
```

Figura 9-4. Esempio: programma "di riempimento".

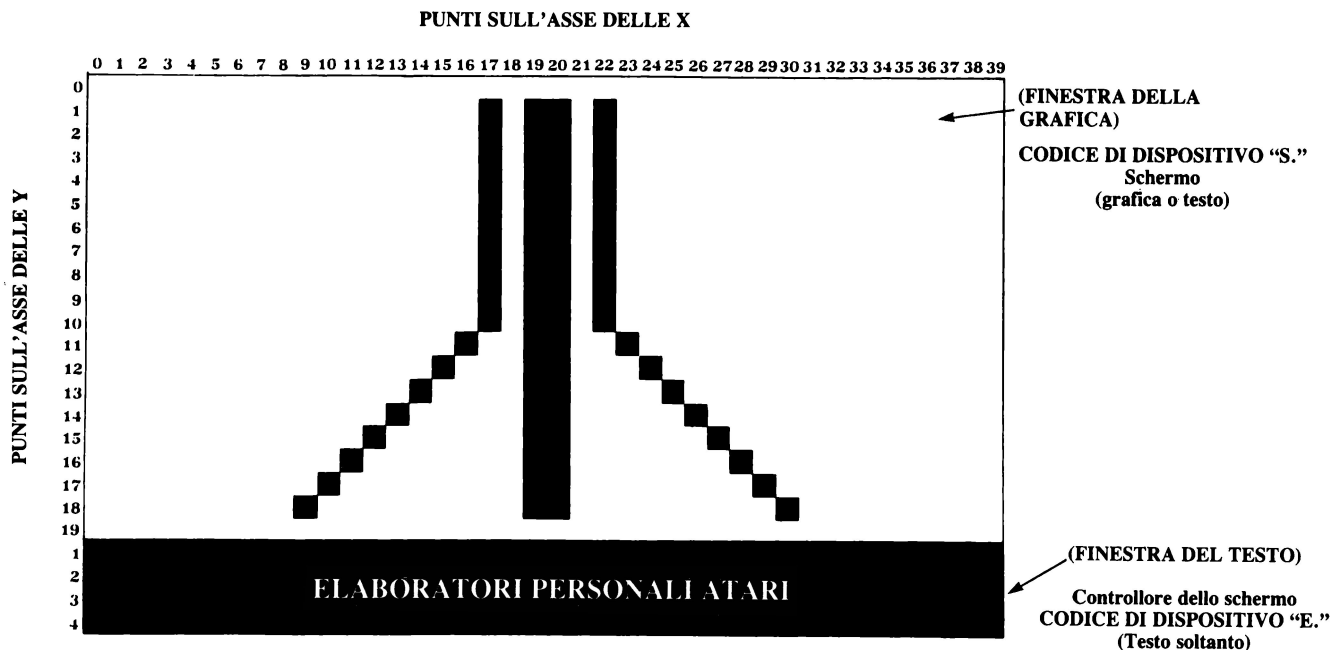



Figura 9-3. Esecuzione del programma "Atari Logo".

Tabella 9-5. MODALITÀ, SETCOLOR, COLORI

Colori per difetto	Modalità o condizione	SETCOLOR (aexp1) Numero del registro di colore	Colore (aexp)	DESCRIZIONE E OSSERVAZIONI
	MODALITÀ 0	0	Il dato	—
BLU CHIARO	e FINESTRA	1	COLOR	Luminosità del carattere (colore eguale allo sfondo)
BLU SCURO	DEL TESTO A	2	in realtà determina	Sfondo
	SCHERMO INTERO	3	il carattere	—
NERO		4	da stampare	Bordo
ARANCIONE		0	Il dato	Carattere
VERDE CHIARO	MODALITÀ 1	1	COLOR	Carattere
BLU SCURO	e 2	2	in realtà determina	Carattere
ROSSO	(modalità	3	il carattere	Carattere
NERO	di testo)	4	da stampare	Sfondo, bordo
ARANCIONE		0	1	Punto grafico
VERDE CHIARO	MODALITÀ 3, 5	1	2	Punto grafico
BLU SCURO	e 7	2	3	Punto grafico
	(modalità	3	—	—
NERO	a 4 colori)	4	0	Punto grafico (sfondo per difetto, bordo)
ARANCIONE		0	1	Punto grafico
	MODALITÀ 4	1		—
	e 6	2		—
	(modalità	3		—
NERO	a 2 colori)	4		Punto grafico (sfondo per difetto), bordo
VERDE CHIARO		0	—	—
BLU SCURO		1	1	Luminosità del punto grafico (colore eguale allo sfondo)
	MODALITÀ 8	2	0	Punto grafico (sfondo per difetto)
	(1 colore e	3	—	—
NERO	2 luminosità)	4	—	Bordo

Tabella 9-6. INSIEME DEI CARATTERI INTERNI

Colonna 1				Colonna 2				Colonna 3				Colonna 4			
#	CAR	#	CAR	#	CAR	#	CAR	#	CAR	#	CAR	#	CAR	#	CAR
0	Space	16	0	32	@	48	P	64		80		96		112	p
1	!	17	1	33	A	49	Q	65		81		97	a	113	q
2	”	18	2	34	B	50	R	66		82		98	b	114	r
3	#	19	3	35	C	51	S	67		83		99	c	115	s
4	$	20	4	36	D	52	T	68		84		100	d	116	t
5	%	21	5	37	E	53	U	69		85		101	e	117	u
6	&	22	6	38	F	54	V	70		86		102	f	118	v
7	’	23	7	39	G	55	W	71		87		103	g	119	w
8	(	24	8	40	H	56	X	72		88		104	h	120	x
9	)	25	9	41	I	57	Y	73		89		105	i	121	y
10	*	26	:	42	J	58	Z	74		90		106	j	122	z
11	+	27	;	43	K	59	[	75		91	①	107	k	123	
12	,	28	<	44	L	60	\	76		92		108	l	124	|
13	—	29	=	45	M	61	]	77		93		109	m	125	①
14	–	30	>	46	N	62	∧	78		94		110	n	126	①
15	/	31	?	47	O	63	—	79		95		111	o	127	①

1. Nella modalità θ questi caratteri, per essere stampati, devono essere preceduti da un ESCAPE, CHR$(27).

Tabella 9.7 - ASSEGNAZIONE CARATTERE/COLORE					
		Conversione 1	**Conversione 2**	**Conversione 3**	**Conversione 4**
MODALITÀ 0	REGISTRO 2	# + 32	# + 32	# – 32	NESSUNA
		POKE 756,224		POKE 756,226	
MODALITÀ 1 O MODALITÀ 2	REGISTRO 0	# – 32	# + 32	# – 32	# – 32
	REGISTRO 1	NESSUNA	# + 64	# – 64	NESSUNA
	REGISTRO 2	# + 160	# + 160	# + 96	# + 96
	REGISTRO 3	# + 128	# + 192	# + 64	# + 128

Assegnazione di colori ai caratteri, nelle modalità di testo 1 e 2.

Descriveremo qui il procedimento che permette di assegnare vari colori all'insieme dei caratteri Atari. Consultare innanzitutto la tabella 9-6 per trovare il numero del carattere, poi la tabella 9-7, per stabilire la conversione del numero richiesta per assegnare ad esso il registro di colore voluto.

Ad esempio, per assegnare il registro di colore 0, nella modalità 2, alla lettera minuscola "r", cioè per fare in modo che il suo colore sia quello contenuto nel registro 0:

1. Cercare nella tabella 9-6 la colonna ed il numero per "r" (colonna 4, numero 114);
2. Nella tabella 9-7, individuare la colonna 4. La conversione è il numero del carattere meno 32 (114 – 32 = 82).
3. Per mezzo di un comando POKE, immettete nell'indirizzo base dei caratteri (CHBAS) il dato 226, per specificare che volete una lettera minuscola o uno dei simboli grafici speciali; per esempio:

   ```
   POKE 756,226
       oppure
   CHBAS = 756
   POKE CHBAS,226
   ```

   Per tornare alle lettere maiuscole, ai numeri e ai segni di punteggiatura, immettere 224 nel CHBAS con un comando POKE.

4. Il comando PRINT, con il numero convertito (82), assegna alla "r" minuscola il registro 0 nella modalità 2 (vedere la tabella 9-5).

Simboli grafici impostabili dalla tastiera

Sono i caratteri che appaiono sul video quando si preme il tasto **[CTRL]** assieme ad uno dei tasti alfabetici; il quadro dei caratteri grafici è illustrato sul retrocopertina. Si possono usare questi caratteri per la grafica, i disegni, etc., nella modalità 0 e, se è stato cambiato il CHBAS, anche nelle modalità 1 e 2

SUONI E DISPOSITIVI DI CONTROLLO PER I GIOCHI

In questo capitolo descriveremo le istruzioni che permettono di generare suoni e note musicali per mezzo del sistema audio del monitor televisivo. Si possono emettere, simultaneamente, fino a quattro suoni differenti e creare così un'armonia. Potrete usare l'istruzione SOUND anche per simulare esplosioni, fischi, e per dare altri interessanti effetti sonori. Nella parte rimanente di questo capitolo descriveremo alcune funzioni riguardanti i controllori a tastiera, a leva e a manopola, che si possono collegare alla console ed utilizzare, ad esempio, nei programmi BASIC per i giochi.

I comandi e le funzioni di cui parleremo in questo capitolo sono:

SOUND	**PADDLE**	**STICK**
	PTRIG	**STRIG**

SOUND (SO.)

Formato: SOUND aexp1, aexp2, aexp3, aexp4

Esempio: 100 SOUND 2, 204, 10, 12

Non appena viene eseguita un'istruzione SOUND, l'audio del televisore inizia ad emettere la nota specificata, nell'istruzione stessa, dell'aexp2. La nota continuerà a suonare finché il programma non incontra un'altra istruzione SOUND con la stessa aexp1 (voce) o un comando END. Il comando SOUND si può usare sia in modo diretto che in modo differito.

I parametri di SOUND determinano i seguenti elementi del suono:

aexp1 = *Voce.* Come abbiamo detto, il BASIC Atari prevede la possibilità di quattro voci simultanee. L'aexp1 indica quale delle quattro voci dovrà eseguire la nota specificata nella stessa istruzione; essa può dunque variare da 0 a 3. È inoltre da tener presente che ogni voce richiede un'istruzione SOUND a parte.

aexp2 = *Tono.* Può essere un qualsiasi numero compreso tra 0 e 255. Il tono è tanto più basso quanto più grande è il valore dell'aexp2. La tabella 10-1 definisce i numeri di tono per le note musicali che sono comprese nelle due ottave sopra il DO di mezzo o nell'ottava sotto al DO di mezzo.

aexp3 =	*Distorsione*. Può essere un numero pari compreso tra 0 e 14. Si usa per creare effetti sonori particolari. 10 dà un tono "puro", mentre 12 dà un suono tremolante. Si può produrre una specie di ronzio, come il rumore di un motore in lontananza, utilizzando due comandi SOUND alternati con valore di distorsione (aexp3) 0 e 1, rispettivamente. Il valore 1 forza l'emissione dei suoni dall'altoparlante, ed essi usciranno al volume specificato dall'aexp4. Gli altri numeri servono a dare effetti speciali o rumori, e per uso sperimentale.
aexp4 =	*Volume*. Può andare da 1 a 15. Se il valore dell'aexp4 è 1, si ottiene un suono appena udibile, mentre il volume 15 è molto alto. Il valore 8 è considerato normale. Se si danno diverse istruzioni SOUND, per ascoltare simultaneamente più voci, il volume totale (la somma delle aexp4 delle varie istruzioni) non dovrebbe essere superiore a 32. Altrimenti si ottiene un suono decisamente spiacevole.

Table 10-1. Valori del tono per le note musicali

NOTE ALTE	DO	29
	SI	31
	LA # o SI b	33
	LA	35
	SOL # o LA b	37
	SOL	40
	FA # o SOL b	42
	FA	45
	MI	47
	RE # o MI b	50
	RE	53
	DO # o RE b	57
	DO	60
	SI	64
	LA # o SI b	68
	LA	72
	SOL # o LA b	76
	SOL	81
	FA # o SOL b	85
	FA	91
	MI	96
	RE # o MI b	102
	RE	108
	DO # o RE b	114
DO DI MEZZO	DO	121
	SI	128
	LA # o SI b	136
	LA	144
	SOL # o LA b	153
	SOL	162
	FA # o SOL b	173
	FA	182

NOTE BASSE	MI	193
	RE # o MI b	204
	RE	217
	DO # o RE b	230
	DO	243

Il programma seguente dimostra come si possono utilizzare i valori per le note della tabella 10-1 per suonare la scala di DO.

```
10 READ A
20 IF A=256 THEN END
30 SOUND 0,A,10,10
40 FOR W=1 TO 400:NEXT W
50 PRINT A
60 GOTO 10
70 END
80 DATA 29,31,35,40,45,47,53,60,64,72,81,91,96,108,121
90 DATA 128,144,162,182,193,217,243,256
```

Figura 10-1. Programma per suonare una scala musicale.

Notate che l'istruzione DATA, nella riga 80, termina con 256, che non è compreso tra i numeri come valori di tono; esso viene utilizzato come segno di "fine dei dati".

FUNZIONI RIGUARDANTI I DISPOSITIVI DI CONTROLLO PER I GIOCHI

Nella figura 10-2 sono rappresentati due dispositivi di controllo che si possono usare con gli elaboratori personali ATARI. I dispositivi di controllo possono essere collegati direttamente all'elaboratore oppure a dispositivi meccanici esterni, e permettono ad eventi esterni (come la pressione di un pulsante o la posizione di una manopola) di entrare direttamente nell'elaboratore, come dati da elaborare o come mezzi di controllo.

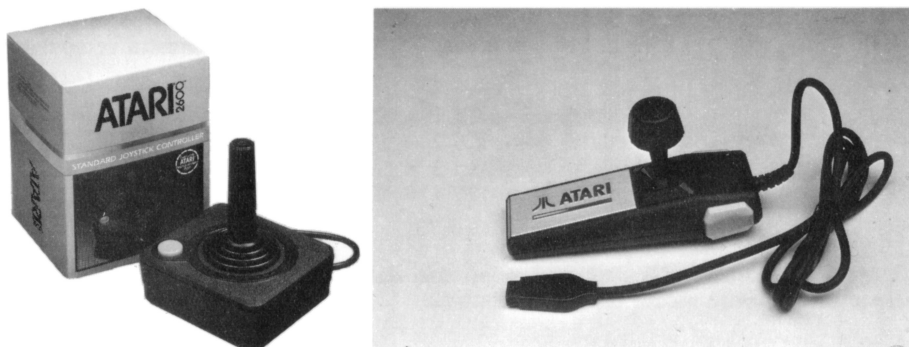



Figura 10-2. Dispositivi di controllo per i giochi.

PADDLE

Formato: PADDLE(aexp)

Esempio: PRINT PADDLE(3)

Questa funzione ha come valore un numero che rappresenta la posizione della manopola ("paddle") di controllo che corrisponde all'aexp. Le manopole di controllo sono numerate da 0 a 7, andando da sinistra a destra. La funzione PADDLE può essere associata ad altre funzioni o comandi per controllare l'esecuzione di una determinata azione, come l'emissione di un suono o il disegno di una configurazione grafica. Per esempio, si può avere l'istruzione **IF PADDLE (3)=14 THEN PRINT "LA MANOPOLA È IN AZIONE".** Il valore che si può ottenere applicando la funzione PADDLE ad un dato argomento è un numero compreso tra 1 e 288: a mano a mano che si ruota la manopola in senso antiorario (verso sinistra), il valore dato da PADDLE aumenta.

PTRIG

Formato: PTRIG(aexp)

Esempio: 100 IF PTRIG(4)=0 THEN PRINT "FUOCO CON I MISSILI!"

LA FUNZIONE PTRIG viene applicata ad un numero che può variare da 0 a 7 ed indica una delle manopole di controllo. Se si sta premendo il pulsante del dispositivo indicato, il valore della funzione è 0; altrimenti è 1.

STICK

Formato: STICK(aexp)

Esempio: 100 PRINT STICK(3)

STICK funziona esattamente come la funzione PADDLE, ma si riferisce alle leve di controllo ("joysticks"). Le leve sono numerate da 0 a 3, andando da sinistra verso destra.

Leva di controllo 1 = STICK(0)
Leva di controllo 2 = STICK(1)
Leva di controllo 3 = STICK(2)
Leva di controllo 4 = STICK(3)

La figura 10-3 mostra quali saranno i valori dati dalla funzione STICK a seconda della direzione in cui viene mossa la leva di controllo.

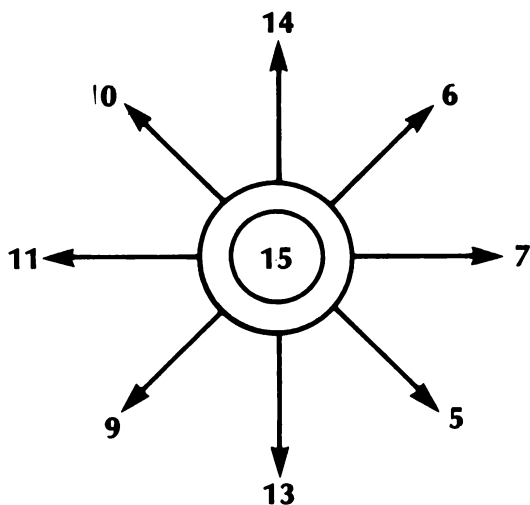



Figura 10-3. Movimenti delle leve di controllo.

STRIG

Formato: STRIG(aexp)

Esempio: 100 IF STRIG(3)=0 THEN PRINT "FUOCO CON UN SILURO"

La funzione STRIG opera esattamente come la funzione PTRIG. Si può usare sia con le leve di controllo che con il controllore a tastiera.

CAPITOLO 11

TECNICHE DI PROGRAMMAZIONE AVANZATA

In questo capitolo accenneremo alle possibilità di incrementare l'efficienza della programmazione, di risparmiare spazio di memoria e di combinare programmi in linguaggio macchina con programmi in BASIC Atari. Questo capitolo non comprende le istruzioni riguardanti il chip microprocessore 6502, né quelle per la programmazione in linguaggio macchina. Si raccomanda quindi di acquistare la cartuccia Atari Assembler Editor e di studiare attentamente l'"Assembler Editor Manual" dell'Atari.

SUGGERIMENTI PER RISPARMIARE MEMORIA

Accenneremo qui ai metodi cui si può ricorrere per risparmiare memoria. Alcuni di questi metodi hanno l'inconveniente che i programmi così redatti risultano meno leggibili e più difficili da modificare, ma spesso è necessario ricorrere a tali tecniche per via dei limiti della memoria disponibile.

1. In molti piccoli elaboratori, l'eliminazione di spazi vuoti tra parole e caratteri quando si scrive sulla tastiera può far risparmiare spazio di memoria. Questo non è vero per il sistema di elaborazione personale ATARI, perché gli spazi superflui vengono rimossi automaticamente. Le istruzioni, nella maggior parte dei casi, non dipendono dal numero di spaziature che appaiono nel programma in entrata. Le uniche spaziature significative per l'elaboratore sono quelle tra due parole chiave successive e tra una parola chiave e un nome di variabile. Per esempio:

   ```
   10 IF A=5 THEN PRINT A
   ```

   Notare la spaziatura tra IF e A, e tra THEN e PRINT. Nella maggior parte dei casi, l'elaboratore interpreterà correttamente anche le istruzioni in cui vengano omesse tutte le spaziature, ma, poiché ciò non sempre è vero, è consigliabile usare il metodo di spaziatura convenzionale.

2. Ogni numero di riga rappresenta l'inizio di ciò che si chiama una "riga logica". In ogni riga logica, 6 bytes di memoria sono occupati dal numero di riga, anche quando la capacità della riga logica non viene poi sfruttata appieno. Se invece sulla stessa riga logica si mettono diverse istruzioni, separandole una dall'altra con i due punti (:), si impiegheranno soltanto 3 bytes in più per ogni istruzione, per codificare l'elemento di separazione.

   Quindi, se vi è necessario risparmiare memoria evitate di strutturare un programma in modo simile a quello seguente:

   ```
   10 X=Y+1
   20 Y=Y+1
   30 Z=X+Y
   40 PRINT Z
   50 GOTO 10
   ```

Concentrate invece le righe:

```
10 X=X+1:Y=Y+1:Z=X+Y:PRINT Z:GOTO 10
```

E risparmierete così 12 bytes.

3. Anche l'uso di variabili e costanti può servire a utilizzare meglio lo spazio di memoria. Ogni volta che si usa una costante (4, 5, 16, 3.14159, etc.) si utilizzano 7 bytes. La definizione di una nuova variabile richiede 8 bytes più la lunghezza del nome della variabile (in caratteri), ma ogni successiva utilizzazione di una variabile già definita occupa un solo byte, quale che sia la lunghezza del nome della variabile. Perciò, se in un programma una stessa costante (come 3.14159) viene utilizzata più di una volta o due, conviene assegnare il valore della costante ad una variabile ed utilizzare quest'ultima in tutto il programma. Per esempio:

```
10 PI=3.14159
20 PRINT "L'AREA DI UN CERCHIO È EGUALE AL RAGGIO
AL QUADRATO PER ";PI
```

4. Le stringhe richiedono 2 bytes di intestazione ed 1 byte per ogni carattere (inclusi gli spazi).
5. Per definire una variabile stringa, sono necessari 9 bytes più la lunghezza del nome della variabile, più lo spazio occupato dall'istruzione DIM, più la lunghezza della stringa stessa (un byte per carattere, inclusi gli spazi). Ovviamente, l'uso di variabili stringa è molto costoso, in termini di RAM.
6. La definizione di una matrice richiede 15 bytes, più la lunghezza del nome della variabile matrice, più lo spazio occupato dall'istruzione DIM, più 6 volte la misura della matrice stessa (prodotto del numero di righe e del numero di colonne). Perciò, una matrice di 25 righe per 4 colonne richiederebbe 15 bytes +3 bytes circa (per il nome della variabile) + 10 bytes circa (per l'istruzione DIM) + 6 per 100 bytes (la misura della matrice), cioè circa 630 bytes.
7. Ogni carattere di un'istruzione REM (escluse le lettere REM stesse) occupa un byte di memoria. Note ed osservazioni sono utili per comprendere meglio un programma, ma talvolta è necessario rimuoverle per risparmiare spazio di memoria.
8. Le subroutines possono servire a risparmiare memoria, perché una subroutine assieme a varie chiamate brevi occupa un minore spazio di memoria rispetto alla duplicazione ripetuta di una stessa serie di istruzioni. D'altra parte, una subroutine che venga chiamata solo una volta occupa dei bytes in più per le istruzioni GOSUB e RETURN.
9. Le parentesi occupano un byte ciascuna. In alcuni casi, può essere una buona idea mettere delle parentesi in più per rendere più comprensibile al programmatore una data espressione. Tuttavia si può risparmiare qualche byte di memoria eliminando le parentesi superflue e sfruttando le regole di precedenza tra gli operatori (vedi il capitolo 1).

PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

Il linguaggio macchina è scritto tutto in codice binario. L'elaboratore personale ATARI contiene un microprocessore 6502 ed è possibile chiamare dal BASIC, usando la funzione USR, delle subroutine in codice macchina 6502. In un programma si possono dunque

immettere brevi routines compiendo l'assemblaggio manualmente (se necessario).

Prima di tornare al BASIC, la routine in linguaggio Assembler deve eseguire un'istruzione di pull dell'accumulatore (**PLA**) per rimuovere dallo stack il numero (N) di argomenti dell'input. Se questo numero è diverso da 0, allora tutti gli argomenti dell'input devono essere tolti dallo stack, usando sempre PLA (vedere la figura 6-1).

La subroutine dovrebbe terminare mettendo il byte meno significativo del suo risultato nella locazione 212 (decimale) e tornando poi al BASIC per mezzo di un'istruzione RTS (Ritorno da una Subrotine). L'interprete BASIC convertirà il numero binario di 2-byte immagazzinato nelle locazioni 212 e 213 in un numero intero compreso tra 0 e 65535 nel formato in virgola mobile, per ottenere il valore della USR.

Si può usare la funzione ADR per passare i dati immagazzinati in vettori o stringhe ad una subroutine in linguaggio macchina. Usate la funzione ADR per ottenere l'indirizzo del vettore o della stringa e usate poi questo indirizzo come uno degli argomenti di input di USR.

Il seguente programma, Caricatore Esacodice, permette di immettere i dati in codice esadecimale, in quanto esso converte ogni numero esadecimale nel corrispondente numero decimale e conserva il numero decimale in un vettore. Questo viene poi eseguito come una subroutine in linguaggio Assembler (si usa un vettore per conservare uno spazio di memoria per la routine).

```
10 GRAPHICS 0:PRINT "PROGRAMMA CARICATORE
ESACODICE" :PRINT
20 REM IMMAGAZZINA GLI EQUIVALENTI DECIMALI NEL
VETTORE A E LI SCRIVE IN UN'ISTRUZIONE DATA NELLA
RIGA 1500
30 REM DOPO SI DEVE POSIZIONARE IL CURSORE SULLA
RIGA SCRITTA, PREMERE "RETURN" E IMMETTERE IL RESTO
31 REM DEL PROGRAMMA BASIC COMPRESA L'ISTRUZIONE
USR
40 DIM A(50),HEX$(5)
50 REM INGRESSO, CONVERSIONE E IMMAGAZZINAMENTO
DEI DATI
60 N=0:PRINT "IMMETTERE UNO ALLA VOLTA I CODICI
ESADECIMALI, ALLA FINE SCIVERE 'FATTO' ";
70 INPUT HEX$
80 IF HEX$= "FATTO" THEN N=999:GOTO 130
90 FOR I=1 TO LEN(HEX$)
100 IF HEX$(I,I)<="9" THEN N=N*16+VAL(HEX$(I,I)):GOTO 120
110 N=N*16+ASC(HEX$(I,I))-ASC("A")+10
120 NEXT I
130 PRINT N:C=C+1
140 A(C)=N
150 IF N<>999 THEN GOTO 60
190 REM STAMPA L'ISTRUZIONE DATA COME RIGA 1500
200 GRAPHICS 0:PRINT "1500 DATA";
```

```
210 C=0
220 C=C+1
230 IF A(C)=999 THEN PRINT "999":STOP
240 PRINT A(C);",";
250 A(C)=0
260 GOTO 220
300 PRINT "METTERE IL NUMERO CORRETTO DI BYTES ALLA
RIGA 1000.": STOP :REM RIGA TRAP
999 REM **MODULO DI ESECUZIONE**
1000 CLR :BYTES=0
1010 TRAP 300:DIM E$(1),E(INT(BYTES/6)+1)
1030 FOR I=1 TO BYTES
1040 READ A:IF A>255 THEN GOTO 1060
1050 POKE ADR(E$)+I,A
1060 NEXT I
1070 REM SEGUE LA PARTE BASIC DEL PROGRAMMA
```

Figura 11-1. Programma di Input del Caricatore Esacodice.

1. Per utilizzare questo programma, innanzitutto immettetelo nell'elaboratore. È poi consigliabile salvarlo su disco o cassetta per poterlo riutilizzare in seguito.

2. Aggiungete la parte del vostro programma in linguaggio BASIC, iniziando dalla riga 1080 e includendo la funzione USR che chiama la subroutine in linguaggio-macchina (vedere l'esempio dato sotto).

3. Contate il numero totale dei codici esadecimali da immettere e, quando viene richiesto, scrivete questo numero alla riga 1000; se avevate già immesso un altro numero, potete semplicemente sostituirlo.

4. Avviate l'esecuzione del programma ed immettete i codici esadecimali della subroutine a livello macchina, premendo **[RETURN]** dopo ogni immissione. Dopo aver immesso l'ultimo dato, scrivete FATTO e premete **[RETURN]**.

5. Ora sullo schermo appare la riga DATA (1500). Essa non entrerà nel programma finché non spostate il cursore sulla riga DATA e non premete **[RETURN]**.

6. Aggiungete al programma la riga 5 GOTO 1000 per oltrepassare il Caricatore Esacodice (o cancellatelo fino alla riga 260). Salvate ora il programma così completato usando CSAVE o SAVE. È importante far questo prima di eseguire la parte di programma che contiene la chiamata USR. Un errore in una routine in linguaggio macchina può far crollare tutto il sistema. Se ciò dovesse accadere, premete **[RESET]**. Se il sistema ancora non funziona, spegnete l'elaboratore ed accendetelo di nuovo, ricaricate il programma e correggetelo.

Nota: Questo metodo funziona solo con routine in linguaggio macchina che si possano rilocare.

I due programmi esemplificativi che seguono possono essere immessi nel programma

Caricatore Esacodice. Il primo stampa NULLA SI MUOVE mentre il programma macchina cambia i colori. Il secondo fa apparire un struttura grafica, poi cambia i colori.

```
1080 GRAPHICS 1+16
1090 FOR I=1 TO 6
1100 PRINT #6;"nulla si muove!"
1110 PRINT #6;"NULLA SI MUOVE!"
1120 PRINT #6;"nulla si muove!"
1130 PRINT #6;"NULLA SI MUOVE!"
1140 NEXT I
1150 Q=USR(ADR(E$)+1)
1160 FOR I=1 TO 25:NEXT I:GOTO 1150
```

Dopo aver immesso questo programma controllate che la riga 1000 sia:

1000 CLR:BYTES = 21

Scrivete poi RUN e premete **[RETURN]**.

Immettete ora i codici esadecimali, colonna per colonna, come mostra la tabella seguente.

```
68      2
A2      E8
0       E0
AC      3
C4      90
2       F5
BD      8C
C5      C7
2       2
9D      60
C4          BYTES=21
```

Terminata questa operazione, scrivete FATTO e premete **[RETURN]**. Mettete ora il cursore dopo l'ultima entrata (999) sulla riga DATA e premete **[RETURN]**.

Avviate ora l'esecuzione del programma scrivendo GOTO 1000 e premendo **[RETURN]** oppure, se è stata aggiunta la riga 5, scrivendo RUN **[RETURN]**. Premete **[BREAK]** per interrompere il programma e cancellate la riga 5.

Il secondo programma, che segue, dovrebbe essere immesso al posto del programma NULLA SI MUOVE. Controllate che BYTES = ——— sia nella riga 1000. Seguite le istruzioni dei punti 2-6.

```
1080 GRAPHICS 7+16
1090 SETCOLOR 0,9,4
1100 SETCOLOR 1,9,8
1110 SETCOLOR 2,9,4
```

```
1120 CR=1
1130 FOR X=0 TO 159
1140 COLOR INT(CR)
1150 PLOT 80,0
1160 DRAWTO X,95
1170 CR=CR+0.125
1180 IF CR= 4 THEN CR=1
1190 NEXT X
1200 X=USR(ADR(E$)+1)
1210 FOR I=1 TO 15:NEXT I
1220 GOTO 1200
```

Scrivete RUN e premete **[RETURN].**

Immettete i codici esadecimali per questo programma, colonna per colonna.

```
68      2
A2      E8
0       E0
AC      2
C4      90
2       F5
BD      8C
C5      C6
2       2
9D      60
C4        BYTES = 21
```

Quando avete finito, scrivete FATTO e premete **[RETURN]**. Mettete ora il cursore dopo l'ultima entrata (999) sulla riga DATA premette **[RETURN]**.

Avviate l'esecuzione del programma scrivendo GOTO 1000 e premendo **[RETURN]** o aggiungendo la riga 5 GOTO 1000 e scrivendo RUN **[RETURN]**. Premete **[BREAK]** per interrompere il programma e cancellate la riga 5.

La figura 11-2 illustra una subroutine in Assembler utilizzata per far ruotare i colori. È inclusa qui a titolo di informazione.

Subroutine in assembler per far ruotare i colori						
Indirizzo	**Programma oggetto**	**Numero di riga**	**Etichetta**	**Ausili mnemonici**	**Dati**	
		0100				Routine per ruotare i dati COLOR
		0110				da un registro all'altro.
		0120				Fa ruotare 4 colori.
		0130				
		0140				Indirizzo nel sistema operativo
02C4		0150				COLOR 0 = $02C4
02C5		0160				COLOR 1 = $02C5
02C6		0170				COLOR 2 = $02C6
02C7		0175				COLOR 3 = $02C7
		0180				
		0190		*=	$6000	Indirizzo di inizio del progr. macchina*
6000	68	0200		PLA		Cancella dati nello stack (v. POP cap. 4)
6001	A200	0210		LDX	#0	Azzera il registro X
6003	ACC402	0220		LDY	COLOR 0	Salva COLOR 0
6006	BDC502	0230	LOOP	LDA	COLOR 1,X	
6009	9DC402	0240		STA	COLOR 0,X	
600C	E8	0250		INX		Incremento nel registro X (aggiunta di 1)
600D	E002	0260		CPX	#3	Confronta il contenuto del registro X con 2
600F	90F5	0270		BCC	LOOP	Salta se il contenuto di X è minore di 2
6011	8CC602	0280		STY	COLOR 3	Salva COLOR 0 in COLOR 3
6014	60	0290		RTS		Ritorno dalla subroutine
L'Assembler scrive questo		Questa parte è il programma sorgente immesso dal programmatore usando la cartuccia Assembler Atari				

Indica le costanti.
* La routine è rilocabile.
$ Indica un numero esadecimale.

Figura 11-2. Subroutine in Assembler per far ruotare i colori.

PAROLE RISERVATE DEL BASIC

PAROLA RISERVATA:	ABBREVIAZIONI:	BREVE DESCRIZIONE DELL'ISTRUZIONE BASIC
ABS		Funzione che dà il valore assoluto dell'argomento.
ADR		Funzione che dà l'indirizzo di memoria di una stringa.
AND		Operatore logico: l'espressione è vera se entrambe le sottoespressioni congiunte da AND sono vere.
ASC		Funzione che, applicata ad una stringa, dà il numero di codice ATASCII del primo carattere della stringa.
ATN		Funzione che dà l'arco tangente dell'argomento (gradi o radianti).
BYE	**B.**	Uscita dal BASIC e ritorno al sistema operativo residente, o al processore della console.
CLOAD	**CLOA.**	Carica i dati in RAM dal registratore di programmi.
CHR$		Funzione che, applicata ad un valore numerico compreso tra 0 e 255, dà la stringa composta dal carattere ATASCII equivalente a tale valore numerico.
CLOG		Funzione che dà il logaritmo in base 10 dell'argomento.
CLOSE	**CL.**	Istruzione I/O usata per chiudere un file alla conclusione di operazioni I/O.
CLR		È l'opposto di DIM: riinizializza tutte le variabili (dimensionate e non).
COLOR	**C.**	Sceglie il registro di colore.
COM		Come DIM.
CONT	**CON.**	Continua. Riavvia l'esecuzione di un programma interrotto in seguito alla pressione del tasto **[BREAK]** o all'esecuzione di un comando STOP o END; l'esecuzione ricomincia dalla riga successiva a quella dell'interruzione.
COS		Funzione che dà il coseno dell'argomento (gradi o radianti).
CSAVE		Permette la memorizzazione dei programmi residenti in RAM sul registratore a cassetta.
DATA	**D.**	Fa parte della combinazione **READ/DATA.** Serve per identificare i dati che l'istruzione READ deve leggere (gli elementi devono sempre essere separati da virgole).
DEG	**DE.**	L'istruzione **DEG** dice all'elaboratore di eseguire i calcoli delle funzioni trigonometriche in gradi anziché in radianti (in mancanza di quest'istruzione l'elaboratore usa i radianti).

PAROLA RISERVATA:	ABBREVIAZIONI:	BREVE DESCRIZIONE DELL'ISTRUZIONE BASIC
DIM	**DI.**	Riserva la quantità specificata di memoria per matrici, vettori o stringhe, che devono essere tutti dimensionati con l'istruzione DIM, prima di essere definiti o usati in qualsiasi modo.
DOS	**DO.**	Parola riservata per operazioni su disco. Provoca l'apparizione del menu (vedi il Manuale DOS).
DRAWTO	**DR.**	Traccia una linea retta che va dall'ultimo punto tracciato al punto specificato dall'istruzione stessa.
END		Interrompe l'esecuzione di un programma, chiude i files e interrompe i suoni. Si può riprendere l'esecuzione del programma usando **CONT**. (Nota: **END** può essere usato anche più di una volta in un programma).
ENTER	**E.**	Comando I/O usato per conservare dati o programmi in forma non codificata.
EXP		Questa funzione dà il numero e (2.7182818) elevato alla potenza specificata dall'argomento.
FOR	**F.**	Si usa assieme a **NEXT** per instaurare un ciclo **FOR/NEXT.** Determina l'intervallo su cui varierà la variabile di ciclo durante l'esecuzione del ciclo.
FRE		Funzione che dà lo spazio di memoria ancora libero (in bytes).
GET	**GE.**	Usato soprattutto in operazioni su disco per immettere un dato da un singolo byte.
GOSUB	**GOS.**	Salto alla subroutine che inizia al numero di riga specificato da GOSUB.
GOTO	**G.**	Salto incondizionato al numero di riga specificato.
GRAPHICS	**GR.**	Specifica l'uso di una delle nove modalità grafiche. **GR.0.** si può usare per pulire lo schermo.
IF		È usato per provare un salto condizionato o perché venga eseguita un'altra istruzione che è sulla stessa riga soltanto nel caso in cui sia vera la prima espressione.
INPUT	**I.**	Fa si che l'elaboratore richieda l'ingresso di dati dalla tastiera o da un altro dispositivo periferico. L'esecuzione continua solo quando sono stati immessi i dati richiesti ed è stato premuto il tasto **[RETURN]**.
INT		Funzione che dà il più grande numero intero che sia minore o eguale all'argomento. L'arrotondamento è sempre al numero minore, anche quando l'argomento è negativo.
LEN		Funzione che, applicata ad una stringa, dà la lunghezza in bytes di tale stringa, o, equivalentemente, il numero dei suoi caratteri (1 byte contiene 1 carattere).
LET	**LE.**	Assegna un valore ad un dato nome di variabile. Nel BASIC ATARI, LET è opzionale e si può anche omettere.
LIST	**L.**	Scrive il listato del programma sullo schermo o su un altro dispositivo periferico (stampante, registratore).
LOAD	**LO.**	Trasferisce nella memoria dell'elaboratore dati conservati su disco o registratore.

PAROLA RISERVATA	ABBREVIAZIONI:	BREVE DESCRIZIONE DELL'ISTRUZIONE BASIC
LOCATE	**LOC.**	Nelle esecuzioni grafiche, assegna alla variabile specificata il valore contenuto in un dato punto grafico.
LOG		Funzione che dà il logaritmo naturale del suo argomento
LPRINT	**LP.**	Comando diretto alla stampante perché stampi un determinato insieme di dati.
NEW		Cancella tutti i dati attualmente presenti in RAM.
NEXT	**N.**	Istruzione di chiusura di un ciclo **FOR/NEXT.** Tutti i cicli sono eseguiti almeno una volta.
NOT		Inverte il valore logico dell'espressione che lo segue: dà 1 se tale espressione non è vera (ha valore logico 0) e dà valore 0 se l'espressione è vera (ha valore diverso da 0).
NOTE	**NO.**	Vedi Manuale DOS/FMS... Si usa solo per le operazioni su disco.
ON		Si usa con **GOTO** o **GOSUB** per l'esecuzione di un salto. Rende possibili i salti multipli a diversi numeri di riga: dove sarà diretto il salto dipenderà dal valore della variabile o dell'espressione che segue **ON**.
OPEN	**O.**	Apre il file specificato per operazioni di input e output.
OR		Operatore logico che connette due espressioni. Se almeno una di esse è vera, il risultato è 1; se invece sono entrambe false, il risultato è 0.
PADDLE		Funzione che dà la posizione del controllore di gioco a manopola specificato dall'argomento.
PEEK		Funzione che dà il codice decimale del contenuto della locazione di memoria (RAM o ROM) specificata.
PLOT	**PL.**	Traccia un singolo punto alla posizione indicata dalle espressioni che lo seguono (che determinano le coordinate della posizione voluta).
POINT	**P.**	Usato solo per le operazioni su disco.
POKE	**POK**	Inserisce il byte dato nella locazione di memoria specificata. Può modificare soltanto il contenuto delle locazioni RAM. Non tentate di inserire dati nelle locazioni ROM, perché ne risulterebbe un errore.
POP		Rimuove la variabile di ciclo o l'indirizzo di ritorno di un **GOSUB** dalla locazione dello stack in cui è conservata. Si usa quando l'uscita da un ciclo o da una subroutine ha luogo in modo diverso da quello normale.
POSITION	**POS.**	Mette il cursore, sullo schermo, nella posizione specificata.
PRINT	**PR.** o **?**	È un comando I/O: provoca l'uscita di dati dall'elaboratore al dispositivo di output specificato.
PTRIG		Questa funzione dà 0 se si sta premendo il pulsante del controllore di gioco specificato, 1 altrimenti.
PUT	**PU.**	Trasferisce un dato di un singolo byte dall'elaboratore al dispositivo specificato.

PAROLA RISERVATA:	ABBREVIAZIONI:	BREVE DESCRIZIONE DELL'ISTRUZIONE BASIC
RAD		Specifica che i calcoli con le funzioni trigonometriche devono essere eseguite in radianti anziché in gradi. Il valore per difetto è **RAD** (vedi **DEG**).
READ	**REA.**	Legge l'elemento successivo nella lista di **DATA** e lo assegna alla variabile specificata.
REM	**R.** o **[SPACE]**	"Osservazioni" (**REM** sta per "remarks"). Questa istruzione non fa altro che includere commenti del programmatore nel listato del programma, che possano essergli utili per riferimenti futuri. Nessuna istruzione situata su una riga che inizia con **REM** verrà eseguita.
RESTORE	**RES.**	Permette di leggere più di una volta gli elementi di un'istruzione **DATA**, in quanto riporta il puntatore della variabile che deve essere letta all'inizio della lista dei dati.
RETURN	**RET.**	Comanda il ritorno da una subroutine e passa il controllo del programma all'istruzione immediatamente successiva alla riga dell'istruzione **GOSUB.**
RND		Funzione che dà un valore a caso compreso tra 0 e 1, ma mai 1.
RUN	**RU.**	Avvia l'esecuzione di un programma. Pone eguali a 0 le variabili normali ed elimina gli eventuali dimensionamenti di stringhe, vettori e matrici.
SAVE	**S.**	Istruzione I/O che provoca la registrazione di dati o programmi sotto la specificazione di file (filespec) indicata dall'istruzione stessa.
SETCOLOR	**SE.**	Seleziona i dati di colore e luminosità per un particolare registro di colore.
SGN		Funzione che dà 1 se il valore dell'argomento è un numero positivo, 0 se è zero, -1 se è negativo.
SIN		Funzione che dà il seno trigonometrico dell'argomento (**DEG** o **RAD**).
SOUND	**SO.**	Controlla la voce, la distorsione e il volume di un suono o di una nota.
SQR		Funzione che dà la radice quadrata dell'argomento.
STATUS	**ST.**	Chiama la routine di status del dispositivo specificato.
STEP		Si usa con **FOR/NEXT**. Determina la differenza tra ogni coppia di valori della variabile di ciclo.
STICK		Funzione che dà la posizione del controllore di gioco a leva specificato.
STRIG		Funzione che dà 1 se non si sta premendo il pulsante della leva di controllo specificata, 0 altrimenti.
STOP	**STO.**	Interrompe l'esecuzione di un programma, ma non chiude i files né interrompe i suoni.
STR$		Funzione che dà il valore numerico dell'argomento in formato di stringa. Per esempio: **STR$(65)** dà la stringa "65".

PAROLA RISERVATA:	ABBREVIAZIONI:	BREVE DESCRIZIONE DELL'ISTRUZIONE BASIC
THEN		Si usa con **IF:** se l'espressione che segue IF è vera, allora le istruzioni che seguono **THEN** vengono eseguite. Altrimenti il controllo del programma passa alla riga successiva.
TO		Usato con **FOR**, come in "FOR X = 1 TO 10". Separa le espressioni che determinano l'intervallo su cui deve variare la variabile di ciclo.
TRAP	T.	Prende il controllo del programma nel caso di un errore e dirige l'esecuzione verso il numero di riga specificato.
USR		Funzione che dà i risultati di una subroutine in linguaggio macchina.
VAL		Funzione che dà il valore numerico equivalente alla stringa che ha come argomento.
XIO	X.	Istruzione generale di I/O utilizzata per operazioni su disco (vedi il Manuale DOS/FMS) e per operazioni grafiche (ad esempio, per il "riempimento").

APPENDICE B

CODICI DI ERRORE

No. di CODICE di ERRORE	MESSAGGIO DEL CODICE DI ERRORE
2	**Memoria insufficiente** per immagazzinare l'istruzione o il nome della nuova variabile, oppure per dimensionare (DIM) una nuova variabile stringa.
3	**Valore inaccettabile:** un valore atteso come intero positivo è invece negativo, o un valore che doveva essere compreso in un certo intervallo non lo è.
4	**Troppe variabili:** è permesso un massimo di 128 nomi di variabili differenti (vedi capitolo 1).
5	**Errore di lunghezza di una stringa:** si è richiesto un elemento di una stringa il cui indice supera la lunghezza dimensionata.
6	**Dati insufficienti:** l'istruzione READ richiede un numero di dati maggiore di quello fornito dall'istruzione (o istruzioni) DATA.
7	**Numero maggiore di 32767:** il valore non è un intero positivo oppure è maggiore di 32767.
8	**Errore di ingresso:** si è tentato di assegnare (in risposta all'esecuzione di un'istruzione INPUT) un valore non numerico ad una variabile numerica.
9	**Errore nel dimensionamento di una stringa o di un vettore:** la misura data con DIM è maggiore di 32767 o ci si è riferiti ad un vettore o matrice superando la misura dimensionata, oppure il vettore/matrice o la stringa sono già stati DIMensionati; oppure infine ci si è riferiti ad un vettore o a una stringa non dimensionati.
10	**Trabocco dello stack degli argomenti:** ci sono troppi GOSUB o una espressione troppo lunga.
11	**Trabocco nel calcolo con virgola mobile.** Si è tentato di dividere per zero o di riferirsi ad un numero maggiore in modulo o eguale a $1 * 10^{98}$ o minore in modulo di $1 * 10^{-98}$).
12	**Riga non trovata:** un'istruzione GOSUB, GOTO o THEN si riferisce ad un numero di riga inesistente.
13	**Manca la corrispondente istruzione FOR:** l'esecuzione ha incontrato un'istruzione NEXT senza la corrispondente istruzione FOR oppure le istruzioni FOR/NEXT non sono state accoppiate adeguatamente (L'errore viene riportato all'istruzione NEXT e non a FOR).
14	**Riga troppo lunga:** l'informazione è troppo complessa o troppo lunga per essere trattata dal BASIC.
15	**Le istruzioni contenenti GOSUB o FOR sono scomparse:** l'esecuzione ha incontrato un'istruzione NEXT o RETURN mentre le corrispondenti FOR o GOSUB erano state cancellate fin dall'inizio del RUN precedente.
16	**Errore nel RETURN:** l'esecuzione ha incontrato un RETURN senza il GOSUB corrispondente.
17	**Errore durante la pulizia interna:** si è tentato di eseguire una "raccolta della spazzatura", di modificare cioè i bits di RAM scorretti. Questo codice di errore può indicare un problema di hardware, ma può essere anche il risultato di un uso errato di POKE. Provate a scrivere NEW, o a spegnere l'elaboratore, e poi reimmettere il programma senza alcun comando POKE.

No. di CODICE di ERRORE	MESSAGGIO DEL CODICE DI ERRORE
18	**Carattere inammissibile in una stringa:** la stringa non inizia con un carattere corretto; oppure la stringa per un'istruzione VAL non è una stringa numerica.
Nota:	Gli errori che seguono sono errori di INPUT/OUTPUT che derivano dall'uso delle unità disco, delle stampanti o di altre unità periferiche. Ulteriori informazioni relative a questi errori si possono trovare nei manuali o nei fogli di istruzione dei componenti in oggetto.
19	**Il programma da caricare è troppo lungo:** rimane una quantità di memoria insufficiente per completare il caricamento (LOAD).
20	**Il numero di unità periferica è troppo alto:** cioè maggiore di 7; oppure è uguale a 0.
21	**Errore nel caricamento del file:** si tenta di caricare con LOAD un file che non può essere caricato con LOAD (ad esempio un file di dati).
128	**Interruzione da BREAK:** l'operazione è stata interrotta perché l'utente ha premuto il tasto BREAK.
129	**IOCB** già aperto (IOCB sta per Blocco di Controllo dell'Input/Output. Il numero di dispositivo è lo stesso del numero di IOCB).
130	**L'unità in questione non esiste.**
131	**IOCB capace solo di scrivere:** è stato diretto un comando di lettura ad un dispositivo per la sola scrittura (stampante).
132	**Comando non valido** per il dispositivo specifico.
133	**Unità o file non aperto.** Non è stato dato il comando OPEN per il dispositivo che si vorrebbe usare.
134	**Numero di IOCB scorretto:** numero di dispositivo illegittimo.
135	**IOCB capace solo di leggere.** È stato diretto un comando di scrittura ad un dispositivo per la sola lettura.
136	**EOF (Fine del file):** si è arrivati alla fine del file (End of File). **Nota:** Questo messaggio può apparire utilizzando disco o registratore a cassetta.
137	**Record troncato:** tentativo di leggere un record più lungo di 256 caratteri.
138	**Tempo esaurito:** è stato superato il tempo di attesa concesso all'unità periferica. L'unità non risponde.
139	**L'unità periferica non risponde.** Confusione ad una porta seriale o unità disco sbagliata.
140	**Errore di struttura** dell'input nel bus seriale.
141	**Il cursore è fuori dai margini** per una particolare modalità grafica.
142	**Il bus seriale** dei dati è sovraccarico.
143	**Errore di parità nel bus seriale dei dati.**
144	**Errore segnalato dall'unità periferica.** Si è tentato di scrivere su un dischetto fornito di protezione contro le cancellature.
145	**Errore nel confronto** tra lettura e scrittura.
146	**Funzione** non ancora costruita nel gestore.
147	**RAM insufficiente** per il funzionamento della modalità grafica scelta.

No. di CODICE di ERRORE	MESSAGGIO DEL CODICE DI ERRORE
160	**Errore** nel numero dell'unità disco.
161	**Troppi files aperti** (non è disponibile alcun settore della memoria di transito).
162	**Disco pieno** (nessun settore libero).
163	**Errore di sistema** nell'I/O di dati non recuperabile.
164	**Il numero di file non corrisponde:** i collegamenti sul disco sono stati confusi.
165	**Errore** nel nome del file.
166	**Errore di valore nei dati di POINT.**
167	**File protetto.**
168	**Comando illegale** (codice di operazione speciale).
169	**Direttorio pieno** (64 files).
170	**File non trovato.**
171	**POINT non valido.**

INSIEME DEI CARATTERI ATASCII CON RELATIVI INDIRIZZI DECIMALI/ESADECIMALI

Codice decimale	Codice esadecimale	Carattere	Codice decimale	Codice esadecimale	Carattere	Codice decimale	Codice esadecimale	Carattere
0	0		13	D		26	1A	
1	1		14	E		27	1B	
2	2		15	F		28	1C	
3	3		16	10		29	1D	
4	4		17	11		30	1E	
5	5		18	12		31	1F	
6	6		19	13		32	20	Space
7	7		20	14		33	21	!
8	8		21	15		34	22	"
9	9		22	16		35	23	#
10	A		23	17		36	24	$
11	B		24	18		37	25	%
12	C		25	19		38	26	&

CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE
39	27	’	55	37	7	71	47	G
40	28	(	56	38	8	72	48	H
41	29	)	57	39	9	73	49	I
42	2A	*	58	3A		74	4A	J
43	2B	+	59	3B	;	75	4B	K
44	2C	,	60	3C	<	76	4C	L
45	2D	-	61	3D	=	77	4D	M
46	2E	.	62	3E	>	78	4E	N
47	2F	/	63	3F	?	79	4F	O
48	30	0	64	40	@	80	50	P
49	31	1	65	41	A	81	51	Q
50	32	2	66	42	B	82	52	R
51	33	3	67	43	C	83	53	S
52	34	4	68	44	D	84	54	T
53	35	5	69	45	E	85	55	U
54	36	6	70	46	F	86	56	V

CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE
87	57	W	103	67	g	119	77	w
88	58	X	104	68	h	120	78	x
89	59	Y	105	69	i	121	79	y
90	5A	Z	106	6A	j	122	7A	z
91	5B	[	107	6B	k	123	7B	
92	5C	\	108	6C	l	124	7C	|
93	5D	]	109	6D	m	125	7D	
94	5E	^	110	6E	n	126	7E	
95	5F	_	111	6F	o	127	7F	
96	60		112	70	p	128	80	
97	61	a	113	71	q	129	81	
98	62	b	114	72	r	130	82	
99	63	c	115	73	s	131	83	
100	64	d	116	74	t	132	84	
101	65	e	117	75	u	133	85	
102	66	f	118	76	v	134	86	

CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE
135	87		151	97		167	A7	
136	88		152	98		168	A8	
137	89		153	99		169	A9	
138	8A		154	9A		170	AA	
139	8B		155	9B	(EOL) RETURN	171	AB	
140	8C		156	9C	↑	172	AC	
141	8D		157	9D	↓	173	AD	
142	8E		158	9E	←	174	AE	
143	8F		159	9F	→	175	AF	
144	90		160	A0		176	B0	
145	91		161	A1		177	B1	
146	92		162	A2		178	B2	
147	93		163	A3		179	B3	
148	94		164	A4		180	B4	
149	95		165	A5		181	B5	
150	96		166	A6		182	B6	

CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE
183	B7		199	C7		215	D7	
184	B8		200	C8		216	D8	
185	B9		201	C9		217	D9	
186	BA		202	CA		218	DA	
187	BB		203	CB		219	DB	
188	BC		204	CC		220	DC	
189	BD		205	CD		221	DD	
190	BE		206	CE		222	DE	
191	BF		207	CF		223	DF	
192	C0		208	D0		224	E0	
193	C1		209	D1		225	E1	
194	C2		210	D2		226	E2	
195	C3		211	D3		227	E3	
196	C4		212	D4		228	E4	
197	C5		213	D5		229	E5	
198	C6		214	D6		230	E6	

CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE	CODICE DECIMALE	CODICE ESADECIMALE	CARATTERE
231	E7		240	F0		249	F9	
232	E8		241	F1		250	FA	
233	E9		242	F2		251	FB	
234	EA		243	F3		252	FC	
235	EB		244	F4		253	FD	
236	EC		245	F5		254	FE	
237	ED		246	F6		255	FF	
238	EE		247	F7				
239	EF		248	F8				

L'appendice H contiene un programma che esegue la conversione decimale/esadecimale.

Note:

1. ATASCII sta per "ATARI ASCII". Lettere e numeri hanno lo stesso valore che nel codice ASCII, mentre sono differenti i codici di alcuni simboli speciali.

2. Eccetto quando viene esplicitamente indicato, i caratteri da 128 a 255 sono i codici video inversi dei caratteri da 1 a 127.

3. Aggiungete 32 al codice maiuscolo di una lettera per ottenere il codice minuscolo della stessa lettera.

4. Per conoscere il codice ATASCII di un dato carattere, date all'elaboratore il comando (in modo diretto) PRINT ASC("– – –"). Riempite lo spazio vuto con la lettera, il carattere o il numero del quale volete conoscere il codice. E ricordate le virgolette!

5. Nelle pagine 83-85 i simboli speciali sono scritti come appaiono normalmente sul video, cioè bianchi su fondo nero; nelle pagine 86-88 invece abbiamo i simboli video inversi, cioè neri su fondo bianco.

MAPPA DELLA MEMORIA DELL'ATARI XL

Area	Address
	65535
Operating system in ROM	
	55296
Not Used	
	54784
ANTIC registers	
	54272
Input/Output registers	
	54016
POKEY registers	
	53760
Not Used	
	53504
GTIA registers	
	53248
BASIC ROM	
	40960
Cartridge Slot or RAM	
	32768
Screen Memory	
Display List	
BASIC Program and Variables	
	1792
Free RAM	
	1536
Operating system and BASIC Storage	
	0

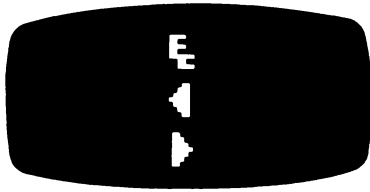
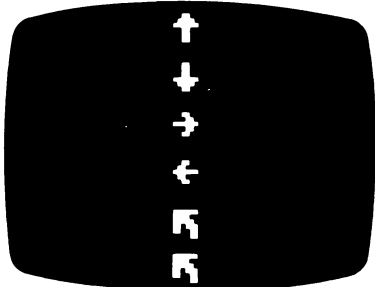
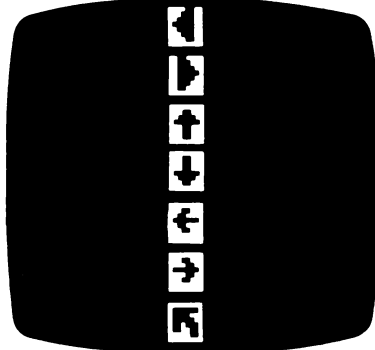
NOTE
Locations 16784 to 32767 are not used in the 600XL.

APPENDICE E

FUNZIONI DERIVATE

Funzioni derivate	Definizione delle funzioni derivate in termini delle funzioni dell'Atari
Secante	SEC(X)=1/COS(X)
Cosecante	CSC(X)=1/SIN(X)
Seno inverso	ARCSIN(X)=ATN(X/SQR(-X*X+1))
Coseno inverso	ARCCOS(X)=-ATN(X/SQR(=X*X+1)+COSTANTE
Secante inversa	ARSEC(X)=ATN(SQR(X*X-1))+(SGN(X-1)*COSTANTE
Cosecante inversa	ARCCSC(X)=ATN(1/SQR(X*X-1))+(SGN(X-1)*COSTANTE
Cotangente inversa	ARCCOT(X)=ANT(X)+COSTANTE
Seno iperbolico	SINH(X)=(EXP(X)-EXP(-X))/2
Coseno iperbolico	COSH(X)=(EXP(X)+EXP(-X))/2
Tangente iperbolica	TANH(X)=-EXP(-X)/(EXP(X)+EXP(-X))*2+1
Secante iperbolica	SECH(X)=2/(EXP(X)+EXP(-X))
Cosecante iperbolica	CSCH(X)=2/(EXP(X)-EXP(-X))
Cotangente iperbolica	COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2=1
Seno iperbolico inverso	ARCSINH(X)=LOG(X+SQR(X*X+1))
Coseno iperbolico inverso	ARCCOSH(X)+LOG(X+SQR(X*X-1))
Tangente iperbolica inversa	ARCTANH(X)=LOG((1+X)/(1-X))/2
Secante iperbolica inversa	ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
Cosecante iperbolica inversa	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Cotangente iperbolica inversa	ARCCOTH(X) =LOG((X+1)/(X-1))/2

Note:

1. Se si stanno eseguendo i calcoli in radianti (se cioè si è nel modo per difetto, RAD), la costante = 1.57079633. Se si è invece in DEG, la costante = 90.
2. In questa tavola, la variabile X inclusa tra parentesi rappresenta il valore o l'espressione su cui deve operare la funzione derivata (il suo argomento). Naturalmente per rappresentare l'argomento è ammesso l'uso di qualsiasi nome di variabile.

VERSIONI STAMPATE DEI CARATTERI DI CONTROLLO

I caratteri di controllo dello schermo e del cursore possono comparire come elementi di una stringa per essere così utilizzati in un programma; oppure potete usarli in modo diretto. Premendo il tasto **[ESC]** prima di introdurre il carattere dalla tastiera, infatti, appariranno sul video i simboli speciali mostrati qui sotto (Vedi capitolo 1 - Tasto **[ESC]**).

APPENDICE G

GLOSSARIO

Alfanumerico:

Le lettere alfabetiche A-Z, i numeri 0-9 ed alcuni simboli (esclusi i segni di punteggiatura e i simboli grafici).

ATASCII:

Sta per Atari American Standard Code for Information Interchange (Codice Americano Standard per l'Interscambio di Informazioni Atari).

BASIC:

Linguaggio di programmazione ad alto livello. Il nome deriva dalle iniziali delle parole componenti l'espressione Beginner's All-purpose Symbolic Instruction Code (codice simbolico per le istruzioni, adatto a tutti gli scopi, per principianti). In BASIC si scrive sempre in lettere maiuscole. È stato sviluppato sotto la guida di J. G. Kemeny e T. E. Kurtz presso il Dartmouth College nel 1963.

Binario:

Sistema numerico in base due. Si hanno perciò soltanto le cifre 0 e 1, che in un elaboratore possono rappresentare i valori vero/falso, oppure acceso/spento, etc.

Bit:

Abbreviazione per cifra binaria (binary digit). Si può dunque pensare ad un bit come alla rappresentazione di "vero o falso ", oppure di un circuito che può essere aperto o chiuso, o di un qualsiasi altro concetto a due valori. Il bit è la più piccola unità di informazione con cui può lavorare un elaboratore.

Bug:

Un errore nel programma o nel software.

Byte:

È generalmente costituito da otto bits (che sono quelli sufficienti per rappresentare il numero decimale 255, o 11111111 in notazione binario). Un byte di informazione può quindi rappresentare qualsiasi carattere ATASCII o numero compreso tra 0 e 255.

Carattere speciale:

Un carattere che l'elaboratore può visualizzare ma che non è una lettera né un numero. I simboli grafici impostabili dalla tastiera dell'Atari, ad esempio, sono caratteri speciali. Lo sono anche i segni di punteggiatura, etc.

Caratteri o simboli grafici impostabili dalla tastiera:

Caratteri prodotti premendo il tasto **[CTRL]** simultaneamente ad altri tasti.

Codice:

Linguaggio comprensibile per l'elaboratore.

Comando:

Ordine diretto all'elaboratore da eseguire immediatamente. Per esempio, il comando BASIC RUN (Vedi Istruzione).

Concatenazione:

Il processo di unione di due o più stringhe per formarne una più lunga.

Controllo dello schermo (Editing):

Modificare o correggere un programma o dei dati che appaiono sul video per poi trasferirli alla memoria dell'elaboratore.

Copia solida (Hard Copy):

Stampa dell'output, in opposizione alla visualizzazione temporanea sul monitor televisivo.

CPU (Central Processing Unit, o unità centrale di elaborazione):

L'unità principale dei microelaboratori come i sistemi Atari è il cosiddetto microprocessore, o MPU. Un tempo il CPU era quella parte del computer che controllava la memoria e le periferiche. Ora il CPU o il MPU si trova generalmente su un singolo circuito integrato, o "chip" (nel caso dell'Atari un chip microprocessore 6502).

CRT:

Abbreviazione per "tubo a raggi catodici" (usato nel televisore). In pratica, il nome denota spesso il ricevitore televisivo, detto anche "monitor", che funziona come video, per l'output dell'elaboratore.

Cursore:

Quadratino luminoso sul monitor televisivo, che mostra dove apparirà il successivo carattere emesso dalla tastiera.

Dato:

Informazione di qualsiasi tipo.

Debug:

Il processo di individuazione e correzione di errori in un programma.

Difetto:

Una modalità o condizione "assunta" dall'elaboratore finché non gli viene detto di fare in altro modo. Per es., schermo e tastiera saranno i dispositivi per difetto finché non si darà esplicitamente il comando di usare altri dispositivi I/O.

Digitale:

Informazione che può essere rappresentata da una collezione di bits. Virtualmente tutti i moderni elaboratori, specialmente i microelaboratori, usano l'approccio digitale.

Dischetto:

Un piccolo disco. È un mezzo di lettura/scrittura, come il nastro, ma è in forma di disco, conservato e protetto dentro una busta rigida. Il vantaggio del disco rispetto alle cassette o ai nastri per la conservazione di informazioni consiste nel fatto che l'accesso ad una parte qualsiasi del disco è virtualmente immediato. Il Sistema di Elaborazione Personale Atari 800 può controllare simultaneamente fino a quattro unità disco periferiche. In questo manuale usiamo le parole disco e dischetto con lo stesso significato.

DOS:

Abbreviazione per Sistema Operativo su Disco. Software o programma che facilita l'uso di un sistema di unità disco.

Elaboratore:

Qualsiasi dispositivo che possa ricevere e poi eseguire istruzioni di elaborazione di dati. Sia le istruzioni che i dati possono variare da momento a momento. La differenza tra un elaboratore e una calcolatrice programmabile risiede nelle capacità dell'elaboratore di manipolare testi e non soltanto numeri; al contrario, una calcolatrice può operare soltanto su numeri.

Esecuzione:

Fare ciò che specifica un comando o un programma. Eseguire (RUN) un programma o una sua parte.

Espressione:

Una combinazione di variabili, numeri ed operatori, (come +, –, etc.), che si può considerare come un'unità in quanto ha un unico valore specifico. Tale valore può essere una stringa o un numero.

Finestra:

Una parte del video televisivo adibita a compiti particolari; si può avere così una finestra per la grafica ed una per il testo.

Formato:

Specificazione della forma che deve avere un'espressione o altro.

Generatore di numeri a caso:

Può essere l'hardware (come nell'Atari) oppure un programma che fornisce un numero il cui valore è difficile da prevedere. Si usa principalmente nei programmi di giochi, ad esempio per scegliere tra varie possibilità.

Hardware:

L'apparato fisico ed elettronico che costituisce la base di un elaboratore.

Inizializzare:

Assegnare un valore iniziale o di partenza. Nel BASIC Atari, tutte le variabili non vettoriali sono inizializzate a zero quando viene dato un comando RUN. Stringhe e vettori non sono inizializzati.

Input (ingresso o entrata):

Informazione trasferita all'elaboratore. L'output (uscita) è invece l'informazione trasferita dall'elaboratore. In questo manuale usiamo i termini input e output sempre in relazione all'elaboratore.

Interattivo:

Un sistema che risponde rapidamente all'utente. Tutti i sistemi di elaborazione personale sono interattivi.

Interfaccia:

L'elettronica usata per permettere a due dispositivi di comunicare.

IOCB:

Blocco di controllo dell'input/output. Un blocco di dati conservati in RAM che rappresentano per il sistema operativo le informazioni di cui ha bisogno per le operazioni di I/O.

I/O:

Abbreviazione per input/output (ingresso/uscita). I dispositivi I/O comprendono la tastiera, il monitor televisivo, il registratore di programmi, la stampante, le unità disco, etc.

Istruzione:

Un ordine dato all'elaboratore, da eseguire in modo differito (vedi anche Comando). Un'istruzione contiene un numero di riga (che avverte l'elaboratore che l'ordine deve essere eseguito in modo differito), una parola chiave, eventualmente il valore su cui operare, ed il comando **[RETURN]**.

K:

Sta per "Kilo": perciò 1 KByte è uguale a circa 1000 bytes (esattamente 1024 bytes). È anche il codice di dispositivo per la tastiera.

Menu:

Una lista di possibiltà tra cui l'utente può scegliere.

Microelaboratore:

Elaboratore basato su un chip microprocessore; nel caso dell'Atari, il 6502.

Monitor:

Il ricevitore televisivo che l'elaboratore utilizza come video per comunicare con l'utente.

OS:

Abbreviazione per Sistema Operativo (Operating System). È in realtà una collezione di programmi che aiuta l'utente a controllare l'elaboratore.

Output (uscita):

Vedi I/O.

Parallelo:

Due o più cose che accadono simultaneamente. Un'interfaccia parallela, per esempio, controlla un certo numero di segnali elettrici distinti nello stesso tempo. Opposto a seriale.

Parola chiave o parola riservata:

Una parola che ha il significato di istruzione o comando in un linguaggio di elaboratori e che perciò non deve essere usata come nome di variabile o all'inizio di un nome di variabile.

Pixel:

Elemento base della grafica. È essenzialmente un punto sullo schermo, le cui dimensioni dipendono dalla modalità grafica che si sta utilizzando.

Programma:

Una successione di istruzioni che descrivono un procedimento eseguibile dall'elaboratore. Un programma deve essere formulato in un linguaggio che il particolare elaboratore considerato può comprendere.

RAM (Random Access Memory):

Memoria ad Accesso Casuale. Nella maggior parte degli elaboratori è la memoria principale. La memoria RAM serve per conservare dati o programmi.

Regole di precedenza:

Regole che determinano l'ordine in cui devono essere eseguite delle operazioni; riguarda gli operatori aritmetici e logici.

ROM (Read Only Memory):

Memoria di sola lettura. In questo tipo di memoria le informazioni che contiene vengono immesse nella fase di fabbricazione dell'elaboratore e non possono essere modificate dall'utente. I programmi come l'interprete BASIC ed altre cartucce utilizzate con i sistemi Atari utilizzano la memoria ROM.

Salto:

Il BASIC Atari esegue un programma seguendo l'ordine dei numeri di riga. Il programmatore può però modificare tale sequenza di esecuzione, includendo nel programma un'istruzione di saltare un certo numero di righe o di tornare ad una riga precedente. Un salto è allora proprio tale modificazione indotta nella sequenza d'esecuzione.

Salvare:

Copiare un programma o un insieme di dati su qualche supporto che non sia la memoria RAM (per esempio su dischetto o nastro).

Schermo:

Lo schermo televisivo. Nel BASIC Atari è un particolare dispositivo I/O, il cui codice è "S:".

Seriale:

Opposto a parallelo. Se un insieme di operazioni è svolto in modo seriale, significa che le operazioni sono eseguite una alla volta, in successione. Si può avere ad esempio un'interfaccia seriale.

Simbolo di "pronto":

Un simbolo che appare sullo schermo del monitor ed indica che il computer è pronto ad accettare l'input dalla tastiera. Nel BASIC Atari, ha la forma della parola "READY". Il BASIC Atari usa inoltre un punto interrogativo, "?", per avvertire l'utente che deve immettere informazioni o eseguire altre azioni.

Software:

Opposto ad hardware. Si riferisce a programmi e a dati.

Stringa:

Una successione di lettere, numeri ed altri caratteri. Può essere conservata in una variabile stringa. Ogni nome di variabile stringa deve terminare con $.

Stringa vuota:

Stringa che non contiene alcun carattere.

Subroutine:

Una parte di un programma, che, in BASIC, viene eseguita in seguito ad una istruzione speciale (GOSUB): questa possibilità fornisce ad una singola istruzione la potenza di tutto un programma. La subroutine è un costrutto molto potente.

Variabile:

Si può pensare ad una variabile come ad una scatola che possa conservare un valore. I valori assegnati ad una variabile possono essere numeri o stringhe.

Vettore:

Una lista di valori numerici conservati in locazioni di memoria adiacenti, riservate a tale scopo da un'istruzione DIM. Ci si può riferire ad un vettore per mezzo di una variabile vettore ed ai suoi elementi individuali per mezzo di variabili sottoscritte.

APPENDICE H

PROGRAMMI TIPO PER L'UTENTE

Quest'appendice contiene alcuni programmi e routines che mostrano cosa si può fare con il sistema di elaborazione personale ATARI. Includiamo qui anche un programma di conversione decimale/esadecimale, che può essere utile per chi voglia scrivere programmi che richiedono questo tipo di conversione.

PROGRAMMA A BILANCIO DEL CONTO CORRENTE

Questo è uno dei programmi "tradizionali", che può scrivere ogni principiante. L'utente introduce i dati riguardanti gli assegni saldati o non ancora saldati ed i versamenti accreditati o non accreditati.

```
10 DIM A$(30)
11 DIM MSG$(50),MSG1$(40),MSG2$(40)
12 DIM MSG3$(40),MSG4$(40),MSG5$(40)
13 DIM MSG6$(50)
20 SALDO=0
30 GRAPHICS 0:? :? "BILANCIO DEL TUO CONTO CORRENTE"
40 ? :? "Potete correggere i dati in qualsiasi momento introducendo valori
negativi per le lire"
50 MSG1$="VECCHIO ASSEGNO (NON PAGATO)"
60 MSG2$="VERSAMENTO NON ACCREDITATO"
70 MSG3$="VECCHIO ASSEGNO PAGATO"
80 MSG4$="VERSAMENTO ACCREDITATO"
90 MSG5$="NUOVO ASSEGNO (COMPETENZE BANCA)"
95 ?
100 MSG6$="NUOVO VERSAMENTO (INTERESSI MAT.)"
150 TRAP 150:? "Scrivi il bilancio iniziale che risulta dal tuo libretto di
assegni":INPUT VSBIL
160 TRAP 160:? :? "Introduci il bilancio iniziale indicato dalla
banca";:INPUT BANKBIL
165 TRAP 40000
170 GOTO 190
180 CLOSE # 1:GRAPHICS 0:? "LA STAMPANTE NON È IN
GRADO DI FUNZIONARE"
185 ? :? "PER FAVORE VERIFICA I CONTATTI"
190 PERM=0
200 ? :? "Vuoi registrare i dati sulla stampante";:INPUT A$
210 REM
```

```
211 REM CONTROLLO DELLA RISPOSTA
212 REM
213 IF LEN(A$)=0 THEN 200
220 IF A$(1,1)="N"THEN 400
230 IF A$(1,1)<>"S" THEN 200
240 TRAP 180
250 REM
251 LPRINT :REM CONTROLLA LA STAMPANTE
252 REM
260 PERM=1
280 LPRINT "IL TUO BILANCIO INIZIALE È DI LIRE";VSBIL
290 LPRINT "IL TUO BILANCIO INIZIALE SECONDO LA BANCA
È DI LIRE ';BANKBIL:LPRINT
400 GRAPHICS 0
401 TRAP 400:? :? "Scegli una delle seguenti azioni:":?
410 ? "(1)";MSG1$
420 ? "(2)";MSG2$
430 ? "(3)";MSG3$
440 ? "(4)";MSG4$
450 3 "(5)";MSG5$
460 ? "(6)";MSG6$
470 ? "(7) FINITO"
480 ?
490 INPUT N:IF N<1 OR N>7 THEN 400
505 TRAP 40000
510 ON N GOSUB 1000,2000,3000,4000,5000,6000,7000
520 MSG$="IL NUOVO BILANCIO È":AMMONT=VSBIL:GOSUB
8000
530 MSG$="IL NUOVO BILANCIO DELL'ESTRATTO CONTO
BANCARIO È DI LIRE":AMMONT=BANKBIL:GOSUB 8000
540 MSG$="ASSEGNI O VERSAMENTI NON
REGISTRATI":AMMONT=SALDI:GOSUB 8000
545 IF PERM THEN LPRINT
550 GOTO 400
1000 REM
1001 REM VECCHIO ASSEGNO (NON PAGATO)
1002 REM
1010 MSG$=MSG1$:GOSUB 8100
1020 SALDI=SALDI+AMMONT.
1030 RETURN
2000 REM
2001 REM VERSAMENTO NON ACCREDITATO
2020 MSG$=MSG2$:GOSUB 8100
2030 RETURN
3000 REM
3001 REM ASSEGNO PAGATO
```

```
3002 REM
3010 MSG$=MSG3$:GOSUB 8100
3020 BANKBIL=BANKBIL-AMMONT
3030 RETURN
4000 REM
4001 REM VERSAMENTO NON ACCREDITATO
4002 REM
4010 MSG$=MSG4$:GOSUB 8100
4020 BANKBIL=BANKBIL+AMMONT
4030 RETURN
5000 REM
5001 REM NUOVO ASSEGNO (COMPETENZE BANCA)
5002 REM
5010 MSG$=MSG5$:GOSUB 8100
5020 VSBIL=VSBIL-AMMONT
5030 ? "IL NUOVO ASSEGNO È ANCORA DA SALDARE";:INPUT
A$
5040 IF LEN(A$)=0 THEN 5060
5050 IF A$(1,1)<>"N" THEN 5060
5055 BANKBIL=BANKBIL-AMMONT
5057 IF PERM THEN LPRINT "IL CONTO È SALDATO"
5058 RETURN
5060 IF A$(1,1)<>"S" THEN 5030
5070 SALDI=SALDI+AMMONT
5075 IF PERN THEN LPRINT "IL CONTO È ANCORA DA
SALDARE"
5080 RETURN
6000 REM
6001 REM NUOVO VERSAMENTO (INTERESSI)
6002 REM
6010 MSG$=MSG6$:GOSUB 8100
6020 VSBIL=VSBIL+AMMONT
6030 ? "IL VOSTRO NUOVO VERSAMENTO È STATO
ACCREDITATO";:INPUT A$
6040 IF LEN(A$)=0 THEN 6030
6050 IF A$(1,1)<>"S" THEN 6060
6052 BANKBIL=BANKBIL+AMMONT
6053 IF PERM THEN LPRINT "IL VERSAMENTO È STATO
ACCREDITATO"
6055 RETURN
6060 IF A$(1,1)<>"N" THEN 6030
6070 SALDI=SALDI-AMMONT
6075 IF PERM THEN LPRINT "IL VERSAMENTO NON È STATO
ACCREDITATO"
6080 RETURN
7000 REM
```

```
7001 REM
7002 GRAPHICS 0
7003 ? "IL BILANCIO DELLA BANCA MENO (ASSEGNI NON
PAGATI - VERSAMENTI NON REGISTRATI) DOVREBBE
ESSERE";
7004 ? "ORA UGUALE AL TUO BILANCIO"
7020 DIF=VSBIL-(BANKBIL-SALDI)
7030 IF DIF<>0 THEN 7040
7035 ? "IL BILANCIO FINALE DEL VOSTRO ESTRATTO CONTO È
DI LIRE";BANKBIL:INPUT A$
7036 IF LEN(A$)=0 THEN 7035
7037 IF A$(1,1)="S" THEN ? "CONGRATULAZIONI: I VOSTRI
CONTI TORNANO":END
7038 GOTO 7060
7040 IF DIF>0 THEN ? "IL SALDO DEI VOSTRI CONTI SUPERA DI
LIRE";DIF;"IL SALDO DELLA VOSTRA BANCA"
7041 GOTO 7060
7050 ? "IL VOSTRO SALDO È INFERIORE A QUELLO DELLA
VOSTRA BANCA DI LIRE";-DIF
7060 ? :? "VOLETE FARE DELLE CORREZIONI ?"
7070 ? "(Per fare delle correzioni basta introdurre un valore negativo)"
7080 ? "SCRIVERE SI O NO":INPUT A$
7090 IF LEN(A$)=0 THEN END
7100 IF A$(1,1)="S" THEN RETURN
7110 END
8000 REM
8001 REM ROUTINE PER LA STAMPA DEI MESSAGGI
8002 REM
8010 ? MSG$;"LIRE";AMMONT
8020 IF PERM=1 THEN LPRINT MSG$;"LIRE";AMMONT
8030 RETURN
8100 REM
8101 REM ROUTINE PER LA STAMPA E L'INTRODUZIONE DEI
DATI
8102 REM
8110 TRAP 8110:? "INTRODUCI L'AMMONTARE
DEL";MSG$;:INPUT AMMONT
8120 TRAP 40000
8130 IF PERM=1 THEN LPRINT MSG$;"LIRE";AMMONT
8140 RETURN
```

PROGRAMMA B ORDINAMENTO A BOLLA

Questo programma usa l'operatore di comparazione tra stringhe "<=", che ordina le stringhe secondo i valori ATASCII dei vari caratteri. Poiché il BASIC Atari non possiede vettori di stringhe, tutte le stringhe usate in questo programma sono sottostringhe di una stringa più lunga. Il metodo di "ordinamento a bolla" consiste nel procedimento seguente: ogni stringa viene confrontata con quella successiva e, se la seconda è "minore" della prima, le due stringhe vengono scambiate di posto; ogni coppia di stringhe della lista viene sottoposta a questo confronto e, giunti in fondo alla lista, si ricomincia daccapo. Si scorre in questo modo la lista dall'inizio alla fine, finché non c'è più alcun bisogno di spostare le stringhe perché sono ormai in ordine alfabetico. Il metodo deve il suo nome al fatto che ogni stringa "minore" di quelle che la precedono risale lentamente al posto appropriato, come una bolla d'aria in un tubicino pieno d'acqua. Gli ordinamenti a bolla, sebbene relativamente lenti quando ci sono molti elementi da considerare, sono facili da scrivere, abbastanza brevi e più semplici da comprendere rispetto ad altri.

```
10 DIM B$(1)
20 GRAPHICS 0:? :? "      ORDINAMENTO DI STRINGHE"
30 TRAP 30:? :? "Scrivi la lunghezza massima delle stringhe";:INPUT
SLUN:SLUN1=SLUN-1
31 ? :? "Le stringhe più corte del massimo saranno riempite di bianchi".
35 IF SLUN<1 OR INT(SLUN)<>SLUN THEN ? :? "Voglio un numero
intero positivo":GOTO 30
40 TRAP 40:? :? "Scrivi il numero massimo di stringhe";
42 ? :INPUT ENTRATE
45 IF ENTRATE<2 OR INT(ENTRATE)<>ENTRATE THEN ? :?
"Voglio un numero intero maggiore di 1":GOTO 40
47 TRAP 40000
50 DIM A$(SLUN*ENTRATE),TEMP$(SLUN)
60 ? :? "Adesso scrivete le stringhe, una per volta"
70 ? :? "Quando avete finito premete semplicemente RETURN"
75 ? :? "Vi prego inoltre di avere pazienza mentre preparo la memoria per
eseguire il lavoro...";
80 FOR I=1 TO SLUN*ENTRATE:A$(I,I)=" ":NEXT I
85 ? :?
90 I=1
100 FOR J=1 TO ENTRATE
110 ? "N.";J;"   ";:INPUT TEMP$
120 IF LEN(TEMP$)=0 THEN ENTRATE=J-1:GOTO 190
130 A$(I,I+SLUN1)=TEMP$
140 I=I+SLUN
150 NEXT J
190 ? :? :? "Per favore aspetta mentre rimetto in ordine le stringhe...";
200 GOSUB 1000:REM CHIAMA LA ROUTINE DI SORT
202 ? :?
205 I=1
210 FOR K=1 TO ENTRATE
220 ? "N.";K;"   ";A$(I,I+SLUN1)
```

```
225 I=I+SLUN
230 NEXT K
240 TRAP 300:? :? “Vuoi una copia scritta”;:INPUT B$
250 IF B$(1,1)=“S” THEN 400
300 END
400 I=1:LPRINT :FOR K=1 TO ENTRATE
420 LPRINT “N.”;K;“    ”;A$(I,I+SLUN1)
430 I=I+SLUN:NEXT K:END
1000 REM ROUTINE DI ORDINAMENTO A BOLLA DELLE
STRINGHE
1010 REM Richiede in input: A$,SLUN,ENTRATE
1020 SLUN=SLUN-1:MAX=SLUN*(ENTRATE-1)+1
1040 FOR 1=1 TO MAX STEP SLUN
1050 DONE=1
1060 FOR K=1 TO MAX-I-SLUN1 STEP SLUN
1070 KSLUN1=K+SLUN1:KSLUN=K+SLUN:KSLUNSLUN1
=KSLUN+SLUN1
1080 IF A$(K,KSLUN1)<=A$(KSLUN,KSLUNSLUN1) THEN GOTO
1110
1090 DONE=0
1100 TEMP$=A$(K,KSLUN1):A$(K,KSLUN1)=A$(KSLUN,
KSLUNSLUN1):A$(KSLUN,KSLUNSLUN1)=TEMP$
1110 NEXT K
1120 IF DONE THEN RETURN
1130 NEXT I
1140 RETURN
```

PROGRAMMA C STAMPA DI CARATTERI PER MODALITÀ DI TESTO

Questo programma stampa i caratteri Atari nei loro colori per difetto nelle modalità di testo 0, 1 e 2. Quando immettete questo programma, ricordate che al poto del simbolo per cancellare lo schermo, "↰" scriviamo qui ")".

```
1 DIM A$(1)
5 ? ")":REM PULISCE LO SCHERMO
10. ? "UNA DIMOSTRAZIONE DEI MODI GRAFICI 0,1 e 2"
20 ? "          (MODALITÀ DI TESTO)"
30 ? :? "Per ciascuno dei modi grafici 0, 1 e 2 vedrete apparire sullo
schermo";
31 ? "l'insieme dei caratteri riproducibili da ATARI"
60 ATTESA=1000:REM NUMERO DI RIGA DELLA SUBROUTINE
70 CARBAS=756:REM INDIRIZZO BASE DEI CARATTERI
80 MAIUS=224:REM VALORE PER DIFETTO DI CARBAS
90 MINUS=226:REM LETTERE MINUSCOLE E GRAFICA
95 GOSUB ATTESA
100 FOR L=0 TO 2
112 REM USA IL DISPOSITIVO E: PER IL MODO GRAFICO 0
115 IF L=0 THEN OPEN #1,8,0,"E:":GOTO 118
116 REM USA IL DISPOSITIVO S: PER I MODI GRAFICI 1 E 2
117 OPEN #1,8,0,"S:"
118 GRAPHICS L
120 PRINT "MODO GRAFICO";L
130 FOR J=0 TO 7: REM 8 RIGHE
140 FOR I=0 TO 31: REM 32 CARATTERI PER RIGA
150 K=32*J+1
155 REM NON FA VEDERE "CLEAR SCREEN" E "RETURN"
160 IF K=ASC(")") OR K=155 THEN 180
165 IF L=0 THEN PUT #1,ASC("  "):REM CARATTERE "ESCAPE"
170 PUT #1,K:REM MOSTRA I CARATTERI
180 NEXT I
190 PRINT #1;"  ":REM FINE DELLA RIGA
200 IF L<>2 OR J<>3 THEN 240
210 REM SCHERMO PIENO
220 GOSUB ATTESA
230 PRINT #1;")":REM PULISCE LO SCHERMO
240 NEXT J
250 GOSUB ATTESA
265 PRINT "Caratteri minuscoli e grafici"
270 IF L<>0 THEN POKE CARBAS,MINUS:GOSUB ATTESA
275 CLOSE #1
280 NEXT L
300 GRAPHICS 0:END
```

```
1000 REM ATTENDE IL RETURN
1010 PRINT "Batti RETURN per continuare"
1020 INPUT A
1030 RETURN
```

PROGRAMMA D   EFFETTI DI LUCE

Questo programma illustra un altro aspetto delle possibilità grafiche dell'Atari. Utilizziamo la modalità grafica 7 per ottenere una più alta risoluzione grafica e le istruzioni PLOT e DRAWTO per tracciare le linee del disegno. Alla riga 20 si può scrivere il titolo del programma in caratteri video inversi (usando il tasto Logo Atari) e raffinare così l'effetto dell'esecuzione.

```
10 FOR ST=1 TO 8:GRAPHICS 7
15 POKE 752,1
20 ? :? "        Effetti di luce con ATARI";SETCOLOR 2,0,0
30 SETCOLOR 1,2*ST,8:COLOR 2
40 FOR DR=0 TO 80 STEP ST
50 PLOT 0,0:DRAWTO 100, DR
60 NEXT DR:FOR N=1 TO 800:NEXT N:NEXT ST
70 FOR N=1 TO 2000:NEXT N:GOTO 10
```

PROGRAMMA E   LA BANDIERA AMERICANA

Con questo programma vogliamo mostrare come si possono alternare i colori in una realizzazione grafica, per disegnare, ad esempio, delle strisce. Usiamo la modalità grafica 7 più 16 in modo che tutto lo schermo sia occupato dalla finestra della grafica. Notare la corrispondenza tra istruzioni COLOR e SETCOLOR.

```
10 REM DISEGNO DELLA BANDIERA AMERICANA
20 REM GRAFICA AD ALTA RISOLUZIONE CON 4 COLORI,
SENZA FINESTRA PER IL TESTO
30 GRAPHICS 7+16
40 REM "SETCOLOR 0" CORRISPONDE AL COLORE 1
50 SETCOLOR 0,4,4:ROSSO=1
60 REM "SETCOLOR 1" CORRISPONDE AL COLORE 2
70 SETCOLOR 1,0,14:BIANCO=2
80 REM "SETCOLOR 2" CORRISPONDE AL COLORE 3
90 BLU=3:REM IL BLU È IL COLORE PER DIFETTO
100 REM DISEGNO DI 13 STRISCE ROSSE E BIANCHE
110 C=ROSSO
120 FOR I=0 TO 12
130 COLOR C
140 REM OGNI STRISCIA HA VARIE RIGHE ORIZZONTALI
150 FOR J=0 TO 6
```

```
160 PLOT 0,I*7+J
170 DRAWTO 159,I*7+J
180 NEXT J
190 REM ALTERNA I COLORI
200 C=C+1:IF C>BIANCO THEN C=ROSSO
210 NEXT I
300 REM DISEGNO DEL RETTANGOLO BLU
310 COLOR BLU
320 FOR I=0 TO 48
330 PLOT 0,I
340 DRAWTO 79,I
350 NEXT I
360 REM DISEGNO DELLE 9 FILE DI STELLE BIANCHE
370 COLOR BIANCO
380 K=0:REM INIZIAMO CON UNA FILA DI 6 STELLE
390 FOR I=0 TO 8
395 Y=4+I*5
400 FOR J=0 TO 4:REM FILA DI 5 STELLE
410 X=K+5+J*14:GOSUB 1000
420 NEXT J
430 IF K<>0 THEN K=0:GOTO 470
440 REM AGGIUNGIAMO LA SESTA STELLA A RIGHE ALTERNE
450 X=5+5 * 14:GOSUB 1000
460 K=7
470 NEXT I
500 REM SE VIENE PREMUTO UN TASTO L'ESECUZIONE SI
INTERROMPE
510 IF PEEK(764)=255 THEN 510
515 REM APRIAMO LA FINESTRA DEL TESTO SENZA
CANCELLARE LO SCHERMO
520 GRAPHICS 7+32
525 REM CAMBIAMO DI NUOVO I COLORI
530 SETCOLOR 0,4,4:SETCOLOR 1,0,14
550 STOP
1000 REM DISEGNA UNA STELLA CON CENTRO NEL PUNTO DI
COORDINATE X,Y
1010 PLOT X-1,Y:DRAWTO X+1,Y
1020 PLOT X,Y-1:PLOT X,Y+1
1030 RETURN
```

PROGRAMMA F GABBIANI SULL'OCEANO

In questo programma combiniamo assieme suoni e grafica. I suoni non sono "puri", ma simulano il rumore dell'oceano ed il verso dei gabbiani. Non è però possibile riprodurre sulla stampante i simboli grafici che usiamo per rappresentare i gabbiani. Alla riga 20 assegnate alla variabile stringa indicata da GABB$ la stringa composta dai caratteri che si ottengono premendo **[CTRL]** G, **[CTRL]** F, **[CTRL]** R, **[CTRL]** R, rispettivamente. Nella lista del programma indicheremo tale stringa con "– – – –".

```
10 DIM GABB$(4)
20 GABB$="- - - -"
30 FLAG=1:RIGA=10:COL=10
40 GRAPHICS 1:POKE 756,226:POKE 752,1
50 SETCOLOR 0,0,0:SETCOLOR 1,8,14
60 PRINT #6;"L'oceano"
70 R=INT (RND(0)*11)
80 POSITION 17,17
90 FOR T=0 TO 10
100 SOUND 0,T,8,4
110 FOR A=1 TO 50:NEXT A
120 IF RND(0)>0.8 THEN FOR D=10 TO 5 STEP -1:SOUND
1,0,10,INT (RND(0)*10):NEXT D: SOUND 1,0,0,0
130 GOSUB 200
140 NEXT T
150 FOR T=10 TO 0 STEP -1
160 SOUND 0,T,8,4
170 FOR A=1 TO 50:NEXT A
175 IF RND(0)>0.8 THEN FOR D=10 TO 5 STEP -1:SOUND
1,D,10,8:NEXT D:SOUND 1,0,0,0
180 FOR H=1 TO 10:NEXT H
185 GOSUB 200
190 NEXT T
195 GOTO 70
200 GOSUB 300
210 POSITION COL,RIGA
220 PRINT #6;GABB$(FLAG,FLAG+1)
230 FLAG=FLAG+2:IF FLAG=5 THEN FLAG=1
240 RETURN
300 IF RND(0)>0.5 THEN RETURN
310 POSITION COL,RIGA
320 PRINT #6;"  "
330 A=INT(RND(0)*3)-1
340 B=INT(RND(0)*3)-1
350 RIGA=RIGA+A
360 IF RIGA=0 THEN RIGA=1
370 IF RIGA=20 THEN RIGA=19
380 COL=COL+B
```

```
390 IF COL=0 THEN COL=1
400 IF COL>18 THEN COL=18
420 RETURN
```

PROGRAMMA G VIDEO GRAFFITI

All'esecuzione di questo programma possono partecipare diverse persone: ognuna sarà munita di un controllore a leva, ed ogni leva di controllo sarà associata ad un particolare colore. Manovrandola, i partecipanti possono impostare sullo schermo disegni differenti. Notare l'uso dei comandi STICK e STRIG

```
1 GRAPHICS 0
2 ? "VIDEO GRAFFITI"
5 REM I VETTORI X ED Y CONTENGONO LE COORDINATE
DELLE POSIZIONI DEI PARTECIPANTI (AL MASSIMO 4)
6 REM IL VETTORE COLR CONTIENE I COLORI
10 DIM A$(1),X(3),Y(3),COLR(3)
120 ? "Per disegnare dovete usare le leve di comando"
129 ? :? "Per cambiare i colori dovete usare il pulsante della leva di
comando"
130 ?:? "I colori iniziali sono i seguenti:"
131 ? :? "Leva 1 = rosso"
132 ? "Leva 2 = bianco"
133 ? "Leva 3 = blu"
134 ? "Leva 4 = nero (come lo sfondo)"
135 ? :? "La posizione del nero è indicata da un punto rosso lampeggiante"
136 ? :? "In graphics 8 le leve 1 e 3 sono bianche, mentre la 4 è blu".
138 ? "Quanti sono i partecipanti (da 1 a 4)";
139 INPUT A$:IF LEN(A$)=0 THEN A$="1"
140 LEVEMAX=VAL(A$)-1
145 IF LEVEMAX<0 OR LEVEMAX>=4 THEN 138
147 ? "Modo grafico n. 3=(40 x 24)"
148 ? "Modo grafico n. 5=(80 x 48)"
150 ? "Modo grafico n. 7=(160 x 96)"
151 ? "Modo grafico n. 8=(320 x 192)"
152 INPUT A$:IF LEN (A$)=0 THEN A$="3"
153 A=VAL(A$)
154 IF A=3 THEN XMAX=40:YMAX=24:GOTO 159
155 IF A=5 THEN XMAX=80:YMAX=48:GOTO 159
156 IF A=7 THEN XMAX=160:YMAX=96:GOTO 159
157 IF A=8 THEN XMAX=320:YMAX=192:GOTO 159
158 GOTO 147:REM IL VALORE DI A NON È ACCETTABILE
159 GRAPHICS A+16
160 FOR I=O TO
LEVEMAX:X(I)=XMAX/2+I:Y(I)=YMAX/2+I:NEXT I:REM SI
COMINCIA PIÙ O MENO AL CENTRO DELLO SCHERMO
```

```
161 IF A<>8 THEN 166
162 FOR I=0 TO 2:COLR(I)=1:NEXT I
163 SETCOLOR 1,9,14:REM BLU CHIARO
165 GOTO 180
166 FOR I=0 TO W:COLR(I)=I+1:NEXT I
167 SETCOLOR 0,4,6:REM ROSSO
168 SETCOLOR 1,0,14:REM BIANCO
180 COLR(3)=0
295 FOR J=0 TO 3
300 FOR I=0 TO LEVEMAX:REM CONTROLLIAMO LE LEVE
305 REM CONTROLLIAMO I PULSANTI
310 IF STRIG(I) THEN 321
311 IF A<>8 THEN 320
312 COLR(I)=COLR(I)+1:IF COLR(I)=2 THEN COLR(I)=0:REM
MODALITÀ A DUE COLORI
313 GOTO 321
320 COLR(I)=COLR(I)+1:IF COLR(I)>=4 THEN COLR(I)=0:REM
MODALITÀ A 4 COLORI
321 IF J<0 THEN COLOR COLR(I):GOTO 325
322 IF COLR(I)=0 THEN COLOR 1:GOTO 325
323 COLOR 0:REM IL QUADRATO DI CHI MUOVE LAMPEGGIA
325 PLOT X(I),Y(I)
330 LEVA=STICK(I):REM LEGGE LA LEVA DI CONTROLLO
340 IF LEVA=15 THEN 530:REM NESSUNA MOSSA
342 COLOR COLR(I):REM CONTROLLO DEL COLORE
344 PLOT X(I),Y(I)
350 IF LEVA>=8 THEN 390
360 X(I)=X(I)+1:REM SI SPOSTA A DESTRA
365 REM SE VA OLTRE IL MARGINE SI VA A CAPO
370 IF X(I)>=XMAX THEN X(I)=0
380 GOTO 430
390 IF LEVA>=12 THEN 430
400 X(I)=X(I)-1:REM SI SPOSTA A SINISTRA
410 IF X(I)<0 THEN X(I)=XMAX-1
430 IF LEVA<>5 AND LEVA<>9 AND LEVA<>13 THEN 470
440 Y(I)=Y(I)+1:IF Y(I)=YMAX THEN Y(I)=0:REM SI SPOSTA IN
BASSO
460 GOTO 500
470 IF LEVA<>6 AND LEVA<>10 AND LEVA<>14 THEN 500
480 Y(I)=Y(I)-1:IF Y(I)<0 THEN Y(I)=YMAX-1:REM SI SPOSTA IN
ALTO
500 PLOT X(I),Y(I)
530 NEXT I
535 NEXT J
540 GOTO 295
```

PROGRAMMA H MUSICA CON LA TASTIERA

Questo programma assegna un nota musicale ad ogni tasto della fila superiore della tastiera dell'Atari. Non è possibile però "scrivere" un'armonia, perché i tasti vanno premuti uno alla volta. Il valore musicale dei tasti è il seguente:

TASTO	VALORE MUSICALE
[INSERT]	SI
[CLEAR]	SI b (o LA #)
0	LA
9	LA b (o SOL #)
8	SOL
7	SOL b (o FA #)
6	FA
5	MI
4	MI b (o RE #)
3	RE
2	RE b (o DO #)
1	DO

```
10 DIM ACCORDO(37),MELODIA(12)
20 GRAPHICS 0:? :?"          MUSICA CON LA TASTIERA
25 ? :? "Per suonare una nota premi uno dei tasti 1-9,0,<,>"
26 ? "Prima di premere un nuovo tasto abbandona il vecchio, se non vuoi
un ritardo nell'esecuzione"
30 FOR X=1 TO 37:READ A:ACCORDO(X)=A:NEXT X
40 FOR X=1 TO 12:READ A:MELODIA(X)=A:NEXT X
50 OPEN #1,4,0,"K:"
55 VECCHIO=-1
60 A=PEEK(764):IF A=255 THEN 60
63 IF A=VECCHIO THEN 100
65 VECCHIO=A
70 FOR X=1 TO 12:IF MELODIA(X)=A THEN SOUND
0,ACCORDO(X),10,8:GOTO 100
80 NEXT X
100 I=INT(PEEK(53775)/4):IF (I/2)=INT(I/2) THEN 60
110 POKE 764,255:SOUND 0,0,0,0:VECCHIO=-1:GOTO 60
200 DATA 243, 230, 217, 204, 193, 182, 173, 162, 153, 144, 136, 128, 121,
114, 108, 102, 96, 91, 85, 81, 76, 72, 68, 64, 60
210 DATA 57, 53, 50, 47, 45, 42, 40, 37, 35, 33, 31, 29
220 DATA 31, 30, 26, 24, 29, 27, 51, 53, 48, 50, 54, 55
```

PROGRAMMA I BLUES CON L'ELABORATORE

Questo programma genera note musicali a caso per scrivere delle melodie molto interessanti per bassi programmati.

```
1 GRAPHICS 0:? :? "BLUES CON L'ELABORATORE"
2 PTR=1
3 THNOT=1
5 ACCORDO=1
6 ? "TEMPO DEI BASSI (1=VELOCE)";
7 INPUT TEMPO
8 GRAPHICS 2+16:GOSUB 2000
10 DIM BASE(3,4)
20 DIM LOW(3)
25 DIM LINE(16)
26 DIM JAM(3,7)
30 FOR X=1 TO 3
40 FOR Y=1 TO 4
50 READ A:BASE(X,Y)=A
60 NEXT Y
70 NEXT X
80 FOR X=1 TO 3:READ A:LOW(X)=A
90 NEXT X
95 FOR X=1 TO 16:READ A:LINE(X)=A:NEXT X
96 FOR X=1 TO 3
97 FOR Y=1 TO 7
98 READ A:JAM(X,Y)=A:NEXT Y:NEXT X
100 GOSUB 500
110 T=T+1
115 GOSUB 200
120 GOTO 100
200 REM ELABORA I TONI ALTI
205 IF RND(0)<0.25 THEN RETURN
210 IF RND(0)<0.5 THEN 250
220 NT=NT-1
230 IF NT>7 THEN NT=7
240 GOTO 260
250 NT=NT-1
255 IF NT<1 THEN NT=1
260 SOUND 2,JAM(ACCORDO,NT),10,NT*2
280 RETURN
500 REM ELABORA LA ROBA DI BASE
510 IF BASS=1 THEN 700
520 BDUR=BDUR+1
530 IF BDUR<>TEMPO THEN 535
531 BASS=1:BDUR=0
```

```
535 SOUND 0,LOW(ACCORDO),10,4
550 RETURN
700 SOUND 0,0,0,0
710 SOUND 1,0,0,0
720 BDUR=BDUR+1
730 IF BDUR<>1 THEN 800
740 BDUR=0:BASS=0
750 THNOT=THNOT+1
760 IF THNOT<>5 THEN 800
765 THNOT=1
770 PTR=PTR+1
780 IF PTR=17 THEN PTR=1
790 ACCORDO=LINE(PTR)
800 RETURN
1000 DATA 162,144,121,136,144,108,102,108,108,96,91,96
1010 DATA 243,182,162
1020 DATA 1,1,1,1,2,2,2,2,1,1,1,1,3,2,1,1
1030 DATA 60,50,47,42,40,33,29
1040 DATA 60,50,45,42,40,33,29
1050 DATA 81,68,64,57,53,45,40
2000 PRINT #6:PRINT #6:PRINT #6
2005 PRINT #6;"        Computer"
2006 PRINT #6
2010 PRINT #6;"        Blues"
2030 RETURN
```

PROGRAMMA L PROGRAMMA DI CONVERSIONE DECIMALE/ ESADECIMALE

Quando dovete convertire numeri decimali in numeri esadecimali, o viceversa, potete utilizzare questo programma.

```
10 DIM A$(9),AD$(1)
20 GRAPHICS 0:? :? "          CONVERSIONI ESADECIMALI":?
30 ? :? "Scrivete "D" per la conversione da decimale ad esadecimale"
35 ? :? "Scrivete "E" per la conversione da esadecimale a
decimale":INPUT A$
40 IF LEN(A$)=0 THEN 30
50 IF A$="E" THEN 300
60 IF A$<>"D" THEN 30
90 TRAP 90
100 ? :? "Scrivete un numero decimale compreso tra 0 e 9999999999"
110 ? "DEC:";:INPUT N
120 IF N<0 OR N>=1E+10 THEN GOTO 100
```

```
130 I=9
140 TEM=N:N=INT(N/16)
150 TEMP=TEMP-N*16
160 IF TEMP<10 THEN A$(I,I)=STR$(TEMP):GOTO 180
170 A$(I,I)=CHR$(TEMP-10+ASC("A"))
180 IF N<>0 THEN I=I-1:GOTO 140
190 ? "ESA:";A$(I,9):?
200 GOTO 110
300 TRAP 300
310 PRINT :? "SCRIVETE UN NUMERO ESADECIMALE
COMPRESO TRA 0 E FFFFFFFF"
320  ? "ESA:";:INPUT A$
330 N=0
340 FOR I=1 TO LEN(A$)
345 AD$=A$(I,I):IF AD$<"0" THEN 300
350 IF A$(I,I)<="9" THEN N=N*16+VAL(AD$):GOTO 370
355 IF AD$<"A" THEN 300
357 IF AD$>"F" THEN 300
360 N=N*16+ASC(AD$)-ASC("A")+10
370 NEXT I
380 PRINT "DEC:";N:?
390 GOTO 320
400 END
```

APPENDICE I

LOCAZIONI DI MEMORIA

Nota: Per i programmatori più esperti può essere interessante conoscere le locazioni di memoria che seguono; per questo le abbiamo incluse in questo manuale. I programmatori dell'Atari usano le etichette che riportiamo per le locazioni di memoria, rendendo così più leggibili i programmi.

ETICHETTE	LOCAZIONE DECIMALE	LOCAZIONE ESADECIMALE	COMMENTI E DESCRIZIONE
APPMHI	14,15	DE	La locazione più alta usata dal BASIC (LSB,MSB).
RTCLOK	18,19,20	12,13,14	Contatore di cicli del televisore (1/60 sec.) (LSB, NSB, MSB).
SOUNDR	65	41	Flag di stato del click durante la battitura dei tasti (0 = senza click)
	77		Attract Mode Flag (128=Attract Mode).
LMARGIN, RMARGIN	82,83	52,53	Margine sinistro, destro (difetto: 2, 39).
ROWCRS	84	54	Riga a cui si trova il cursore (finestra della grafica).
COLCRS	85,86	55,56	Colonna in cui si trova il cursore (finestra della grafica).
OLDROW	90	5A	Riga della posizione precedente del cursore (finestra della grafica).
OLDCOL	92,91	5B	Colonna della posizione precedente del cursore (finestra della grafica).
	93	5C	Dato presente alla posizione del cursore (finestra della grafica, eccetto in modalità 0).
NEWROW	96	60	Riga in cui andrà il cursore con un DRAWTO.
NEWCOL	97,98	61,62	Colonna a cui andrà il cursore con un DRAWTO.
RAMTOP	106	6A	Cima attuale della memoria (numero delle pagine).
LOMEN	128,129	80,81	Puntatore del fondo della memoria del BASIC
MEMTOP	144,145	90,91	Puntatore della cima della memoria del BASIC.
STOPLN	186,187	BA,BB	Numero di riga in cui si sono verificati STOP o TRAP (numero binario di due bytes).
ERRSAV	195	C3	Numero di errore.
PTABW	201	C9	Intervallo tra due successive posizioni di stampa (difetto: 10).
FR0	212,213	DA,D5	Bytes meno e più significativi del valore che la funzione USR dà al BASIC.

ETICHETTE	LOCAZIONE DECIMALE	LOCAZIONE ESADECIMALE	COMMENTI E DESCRIZIONE
RADFLG	251	FB	Indicatore RAD/DEG (0=radianti, 6=gradi).
LPENH	564	234	Valore orizzontale della penna luminosa.
LPENV	565	235	Valore verticale della penna luminosa.
TXTROW	656	290	Riga su cui si trova il cursore (finestra del testo).
TXTCOL	657,658	291,292	Colonna su cui si trova il cursore (finestra del testo).
COLOR0	708	204	Registro di colore 0
COLOR1	709	2C5	Registro di colore 1.
COLOR2	710	2C6	Registro di colore 2.
COLOR3	711	2C7	Registro di colore 3.
COLOR4	712	2C8	Registro di colore 4.
MEMTOP	741,742	2E5,2E6	Puntatore della più alta locazione della memoria controllata dall'OS (LSB, MSB).
MEMLO	743,744	2E7,2E8	Puntatore della più bassa locazione della memoria controllata dall'OS
CRSINH	752	2F0	Inibizione del cursore (0= cursore presente, 1= cursore assente).
CHACT	755	2F3	Registro del modo di scrittura dei caratteri (4 = riflesso verticale; 2 = normale; 1 = bianco).
CHBAS	756	2F4	Registro base dei caratteri (difetto: 224) (224 = caratteri maiuscoli, 226 = minuscoli).
ATACHR	763	2FB	Ultimo carattere ATASCII
CH	764	2FC	Ultimo tasto premuto; codice interno (255 cancella il carattere).
FILDAT	765	2FD	Nella grafica, dato per il riempimento di colore (XIO).
DSPFLG	766	2FE	Indicatore della visualizzazione (1 = simboli grafici speciali).
SSFLAG	767	2FF	Segnalatore di partenza/arresto (0= listato normale). Controllato da **[CTRL]** 1.
HATABS	795	31A	Tabella degli indirizzi per il gestore (Handler) (3 bytes/gestore).
IOCB	832	340	Blocchi di controllo dell'I/O (16 bytes/IOCB).
	1664-1791	680-6FE	RAM libera.
CONSOL	53279	D01F	Pulsanti della console (bit 2 = OPTION, bit 1 = SELECT, bit 0 = START. Date POKE 53279, 0 prima di leggere. 0 = tasto premuto).
PACTL	54018	D302	Registro di controllo della porta A (con il registratore di programmi, 52 = acceso, 60 = spento).
PBCTL	54019	D303	Registro di controllo della porta B.
SKCTL	53775	D20F	Registro di controllo della porta seriale. Bit 2=0 (ultimo tasto ancora premuto).

INDICE ANALITICO

INDICE ANALITICO

INDICE ANALITICO

T

U

V

X

TABELLE

MODALITÀ GRAFICHE E FORMATI DELLO SCHERMO

FORMATO DELLO SCHERMO

Modo grafico	Tipo	Colonne	Righe- (schermo diviso)	Righe- (schermo pieno)	Numero colori	Memoria RAM richiesta (bytes) divisa	piena
0	TEXT	40	—	24	1-1/2		992
1	TEXT	20	20	24	5	674	672
2	TEXT	20	10	12	5	424	420
3	GRAPHICS	40	20	24	4	434	432
4	GRAPHICS	80	40	48	2	694	696
5	GRAPHICS	80	40	48	4	1174	1176
6	GRAPHICS	160	80	96	2	2174	2184
7	GRAPHICS	160	80	96	4	4190	4200
8	GRAPHICS	320	160	192	1-1/2	8112	8138
9	GRAPHICS	80	—	192	1		8138
10	GRAPHICS	80	—	192	9		8138
11	GRAPHICS	80	—	192	16		8138
12	GRAPHICS	40	20	24	5	1154	1152
13	GRAPHICS	40	10	12	5	664	660
14	GRAPHICS	160	160	192	2	4270	4296
15	GRAPHICS	160	160	192	4	8112	8138

MODALITÀ, SETCOLOR, COLORI

Colori per difetto	Modalità o condizione	SETCOLOR (aexp1) Numero del registro di colore	Colore (aexp)	DESCRIZIONE E OSSERVAZIONI
	MODALITÀ 0	0	Il dato	—
BLU CHIARO	e FINESTRA	1	COLOR	Luminosità del carattere (colore eguale allo sfondo)
BLU SCURO	DEL TESTO A	2	in realtà determina	Sfondo
	SCHERMO INTERO	3	il carattere	—
NERO		4	da stampare	Bordo
ARANCIONE		0	Il dato	Carattere
VERDE CHIARO	MODALITÀ 1	1	COLOR	Carattere
BLU SCURO	e 2	2	in realtà determina	Carattere
ROSSO	(modalità	3	il carattere	Carattere
NERO	di testo)	4	da stampare	Sfondo, bordo
ARANCIONE		0	1	Punto grafico
VERDE CHIARO	MODALITÀ 3, 5	1	2	Punto grafico
BLU SCURO	e 7	2	3	Punto grafico
	(modalità	3	–	—
NERO	a 4 colori)	4	0	Punto grafico (sfondo per difetto, bordo)
ARANCIONE		0	1	Punto grafico
	MODALITÀ 4	1		—
	e 6	2		—
	(modalità	3		—
NERO	a 2 colori)	4		Punto grafico (sfondo per difetto), bordo
VERDE CHIARO		0	—	—
BLU SCURO		1	1	Luminosità del punto grafico (colore eguale allo sfondo)
	MODALITÀ 8	2	0	Punto grafico (sfondo per difetto)
	(1 colore e	3	—	—
NERO	2 luminosità)	4	—	Bordo

Questo testo, dando per scontato che l'utente abbia letto il libro **"ATARI BASIC - Guida autodidattica"** o un altro equivalente introduttivo al linguaggio BASIC, costituisce una guida di riferimento a programmi, istruzioni, funzioni ed applicazioni speciali del BASIC Atari.

Gli esempi di programmi riportati nel testo hanno la funzione di illustrare nel modo migliore il significato di tutte le funzioni e le parole chiave del linguaggio BASIC senza pretendere di costituire dei modelli di tecnica della programmazione.

Ogni capitolo contiene insiemi di comandi, funzioni o istruzioni che riguardano un aspetto particolare del BASIC Atari. Per esempio, il capitolo 9 contiene tutte le istruzioni sulle capacità grafiche caratteristiche di Atari. Le appendici contengono dei brevi riferimenti ai termini, ai messaggi di errore, alle parole chiave del BASIC, agli indirizzi di memoria e all'insieme dei caratteri ATASCII.

Cod. 9411

L. 16.000

edizioni JCE

ATARI serie XL

a cura di: ATARI Software Support Group